Mário António Rodrigues Grande Abrantes

Mestre em Matemática

Revision Based Total Semantics for Extended Normal Logic Programs

Dissertação para obtenção do Grau de Doutor em Matemática - Lógica e Fundamentos da Matemática

Orientador: Luís Manuel Sancho Moniz Pereira
Professor Catedrático aposentado do Departamento de Informática da FCT-UNL
Co-orientadora: Isabel Oitavem Rocha
Professora Auxiliar do Departamento de Matemática da FCT-UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro de 2013

To Janusz Kusociński, Rudolph Harbig,
and
Dave Wottle

Acknowledgements

First of all, I wish to express my deepest gratitude to my supervisor, Professor Luís Moniz Pereira, for his invaluable support and guidance throughout the preparation of this work. I thank him for his patience and willingness to debate my suggestions and to answer my questions, even when they were messy or not pertinent.

I also thank my co-adviser Professor Isabel Oitavem for her guidance and concern.

I am indebted to Professor Reinhard Kahle that kindly presented me my supervisor, after listening to my intentions to undertake a PhD.

I thank especially Alexandre Pinto, for his willingness for debating my questions and proposals, and for his suggestions and invariably good advices.

I am indebted to the following people who, in one way or another, helped me during this work, answering my questions, commenting my suggestions, sending me requested papers or giving me valuable opinions (the order of the names is not important): Manuel A. Martins, Ofer Arieli, Torsten Schaub, Roland Kaminski, Vladimir Lifschitz, Teodor Przmusinski, Jan Wielemaker, Peter Schueller, Florbela Fernandes, João Leite, Martin J. B.. I especially thank Chiaki Sakama for presenting a paper of which I am co-author at INAP13', and for helpful comments and suggestions.

I acknowledge the *Fundação para a Ciência e Tecnologia do Ministério da Educação e Ciência* and the *Instituto Politécnico de Bragança* for the grant SFRH / PROTEC/ 49747/ 2009.

I acknowledge the *Escola Superior de Tecnologia e de Gestão - IPB* board and the *Departamento de Matemática da Escola Superior de Tecnologia e de Gestão - IPB*, for their interest in providing me with the best conditions to develop my work.

A special thanks to my colleagues Getúlio Igrejas, Mário Escudeiro, Florbela Fernandes, J.P. Coelho, Rui Fernandes and Tiago Pedrosa, for our various, and most of the time funny, conversations about "PhDs".

Last, but for sure not least, I thank with all my heart the love and patience of Ana Rodrigues and of my mother Maria Gentil, during the (perhaps too many) years this work took to be finished.

Abstract

The purpose of this thesis is the development of a formal semantical approach for extended normal logic programs, where contradictions are tackled by means of a reduction *ad absurdum* with respect to default negation mechanism, in the fashion of artificial intelligence belief revision, that leads to a set of implicit revisions. We fulfilled these objectives in two steps. The first one was the implementation of a total paraconsistent models semantics for extended normal logic programs MH_P, that combines the merits of two already existing semantics: it inherits the existence property of the abductive minimal hypotheses semantics MH, a semantics of total models, and the property of detection of support on contradiction of the paraconsistent well-founded semantics with explicit negation $WFSX_P$, a semantics of partial paraconsistent models. As for the second step, we developed a revision procedure for inconsistent constrained theories, that tackles inconsistencies arising from contradictions with respect to explicit negation, i.e., L and $\neg L$ in the same model, and stepping beyond, also tackles inconsistencies arising in more general constrained theories (i.e., theories containing constraints of the type $\perp \leftarrow A, \mathit{not}\ B$, where A is a conjunction of objective literals and $\mathit{not}\ B$ stands for a conjunction of default literals). An algorithm for inconsistency propagation detection was also developed. A characterization of the MH_P with respect to the semantic formal properties of relevance and cumulativity is furnished, by resorting to a set of results that arose from a detailed study of these properties for a class of 2-valued conservative extensions of the stable models semantics.

Keywords: total paraconsistent models, declarative debugging, inconsistency propagation detection, defectivity, excessiveness, irregularity.

Resumo

O objetivo desta tese é o desenvolvimento de uma abordagem formal semântica para programas lógicos normais estendidos (i.e., programas com dois tipos de negação: uma negação explícita e a negação por omissão), na qual as contradições são resolvidas por meio de um mecanismo de raciocínio por redução ao absurdo, ao estilo da revisão de crenças na inteligência artificial. Este objectivo foi alcançado em duas etapas. A primeira delas foi a implementação de uma semântica de modelos totais para programas lógicos estendidos, MH_P, que combina os méritos de duas semânticas já existentes: herda a propriedade de existência da semântica abdutiva de hipóteses mínimas MH, uma semântica de modelos totais para programas normais; herda também a propriedade de detecção de suporte em contradição da semântica paraconsistente $WFSX_P$, uma semântica de modelos paraconsistentes parciais para programas lógicos estendidos. A segunda etapa consistiu no desenvolvimento de um algoritmo de revisão de teorias inconsistentes, que aborda não apenas inconsistências decorrentes de contradições com respeito à negação explícita, ou seja L e $\neg L$ no mesmo modelo, mas também inconsistências com respeito a teorias contendo restrições de integridade mais gerais, do tipo $\perp \leftarrow A, not\ B$, sendo A uma conjunção de literais objectivos e $not\ B$ uma conjunção de literais objectivos negados por omissão. Um algoritmo de detecção de propagação de inconsistências foi também desenvolvido. É apresentada uma caracterização detalhada da MH_P com respeito às propriedades semânticas formais de relevância e cumulatividade, utilizando um conjunto de resultados obtidos de um estudo dessas propriedades para uma classe de extensões conservativas 2-valoradas da semântica de modelos estáveis.

Palavras-chave: modelos totais paraconsistentes, revisão declarativa de programas lógicos, detecção da propagação de inconsistências, defectividade, excesso, irregularidade.

Contents

List of Tables

List of Acronyms, Abbreviations and Nomenclature

ASM	Affix stable models family of semantics
ASM^h	A subclass of ASM
ASM^m	A subclass of ASM
$\hookrightarrow_{bLWFS}$	Reduction system for computing the balanced layered remainder
$bWFSX_P$	Balanced paraconsistent extended well-founded semantics
$CRG(P)$	Complete rule graph of program P
$\triangle$	Operator that takes a normal logic program and retrieves a 3-valued interpretation
$\bigtriangledown$	Operator that translates a well-founded model into a paraconsistent extended well-founded model
Γ	Gelfond-Lifschitz operator
$\mathcal{H}_P$	Herbrand base of program P
$ker_{SEM}(P)$	Semantic kernel of program P
$kerx_{SEM}(P)$	Extended semantic kernel of program P
$\hookrightarrow_{LWFS}$	Reduction system for computing the layered remainder
$\mathcal{MG}_P$	Modules graph of program P
MH	Minimal hypotheses semantics
MH_P	Paraconsistent minimal hypotheses semantics
$P^{\leq T}$	T segment of program P
$Rel_P(a)$	Relevant subprogram of P to atom a
SCC	Strongly connected component
SEM	Abbreviation for "semantics"
$\hookrightarrow_{WFS}$	Reduction system for computing the well-founded model
WFS	Well-founded semantics

$WFSX_P$	Paraconsistent extended well-founded semantics

Chapter 1

Introduction

We make a brief overview of the historical developments in the field of logic programs semantics, that headed to the implementation of 2*-valued conservative extensions of the stable model semantics and of paraconsistent semantics, which are tools we use in this work. Thereon, we justify the interest and succinctly present the essence of the results stated in this thesis.*

Artificial intelligence is about the design of entities (agents) capable of behaving intelligently in some environment. Such an entity needs to access knowledge about this environment, which in turn demands an unambiguous language for expressing this knowledge, together with some way of manipulating sentences of the language in order to infer new knowledge from the knowledge already acquired. Around 1960, McCarthy [Mcc59] first proposed the use of logical formulas as a basis for a knowledge representation language of this type. In his own words,

> Expressing information in declarative sentences is far more modular than expressing it in segments of computer programs or in tables. Sentences can be true in a much wider context than specific programs can be used. The supplier of a fact does not have to understand much about how the receiver functions or how or whether the receiver will use it. The same fact can be used for many purposes, because the logical consequences of collections of facts can be available.

The combination of logic as a representation language, with the theory of automated deduction, led Kowalski and Colmerauer to the creation of the first declarative logic programming language, Prolog [CKRP73, vEK76]. The Prolog language has available the mechanism of *negation as failure*, under which everything that cannot be finitely proven true is concluded false.[1]

[1]This first part of the introduction is a very close adaptation from [BG94].

A number of semantics for normal logic programs, i.e., those with *negation by default*[2], appeared since then, two main of which are the 3-valued well-founded semantics, *WFS* [GRS91], and the 2-valued stable model semantics *SM* [GL88]. The *SM* semantics is generally accepted by the logic programs scientific community as the de facto standard 2-valued semantics. Nevertheless there are advantageous properties the SM semantics lacks, such as model *existence* for every normal logic program, *relevance* and *cumulativity*. Model existence guarantees that every normal logic program has a semantics. This is important to allow arbitrary updates and/or merges involving knowledge bases, possibly from different authors or sources [PP11]. Relevance allows for top-down query solving without the need to always compute complete models, but just the sub-models that sustain the answer to a query, though guaranteed extendable to whole ones [PP11]. As for cumulativity, it allows the programmer to take advantage of tabling techniques [Swi99] for speeding up computations [PP11].

It is thus reasonable to search for 2-valued semantics that *extend conservatively* the *SM*, meaning that for each normal logic program all the stable models are obtained, eventually together with some other models in order to ensure some or all of the referred properties. In [PP11] the authors developed the abductive *minimal hypotheses* semantics for normal logic programs *MH*, which is an existential 2-valued models semantics. Abductive logic programming is an extension of logic programming to perform abductive reasoning [DK02] (see section 3.3). Consider as an example the following program P

$$\begin{aligned} &p \leftarrow not\ p \\ &b \leftarrow a \\ &a \leftarrow \end{aligned}$$

which has no stable models, due to the rule $p \leftarrow not\ p$, but nonetheless has the MH_P model $M = \{a, b, p\}$, which is obtained after adding to P the *abductive explanation* (for M) p, also called *abductive hypothesis*.

A limitation of the semantics for normal logic programs, is that they do not consider the assertion of negation. Several authors [Wag91, Prz90a, GL90, PA92, KS90, GSC98, Prz90b] have stressed the need to endow normal logic programs with a second kind of negation operator, the *explicit negation* "$\neg$", for representing contradictory knowledge, in addition to the default negation operator "*not*", used for representing incomplete information. A language containing an explicit negation operator, has two types of non default literals: positive literals L, and explicitly negated literals $\neg L$, each admitting the corresponding *default* negated version, respectively $not\ L$ and $not\ \neg L$. There are plenty of examples that display the need for explicitly negated literals in logic programming, both in the heads and in the bodies of the rules.

Example 1.1. (adapted from [AP96]) Consider the statement "Penguins do not fly". This statement may be represented within logic programming by $no\_fly(X) \leftarrow penguin(X)$.

[2]*Negation by default* is a semantic counterpart notion for the procedural Prolog notion of *negation as failure*, brought from non-monotonic reasoning into logic programming semantics, as an adaptation of the *closed world assumption* principle, *CWA* [Rei78]. Although both expressions are sometimes used as synonyms, they are not the same (see [Llo87]).

Meanwhile if additionally we wish to represent the statement "Birds fly", $fly(X) \leftarrow birds(X)$, no connection results between the predicates $no\_fly(X)$ and $fly(X)$, although the intention of the programmer is to set them as contradictory (i.e., the predicates are intended not to be both true or both false in a model). In this case it is suitable to have the rule $\neg fly(X) \leftarrow penguin(X)$ instead of $no\_fly(X) \leftarrow penguin(X)$, since a semantics that deals with the operator "$\neg$" will by definition consider predicates $fly(X)$ and $\neg fly(X)$ as contradictory opposites.

Example 1.2. (adapted from [AP96]) Consider the statement "A school bus may cross railway tracks under the condition that there is no approaching train". If we express the statement with the rule $cross \leftarrow not\ train$, then the lack of information about the presence of the train allows the bus to cross the tracks. In this case it is suitable to have instead the rule $cross \leftarrow \neg train$, making $cross$ depend on a proof of $\neg train$ (of course it is the programmer's responsibility to define $\neg train$ in a manner that caters to the need for assuredness of the falsity of $train$).

As a consequence of the need for an explicit negation operator, a number of semantics that interpret this type of operator have been proposed – those for *extended normal logic programs*. Among those are the *paraconsistent* semantics [ADP95, Ari02, BS89, PW89, Sak92], which have the following advantageous property: if SEM is a paraconsistent semantics and P an *inconsistent* extended normal logic program with respect to explicit negation, i.e. a program where all the SEM models have at least a pair of contradictory literals, say b and $\neg b$, then $SEM(P)$ is not mandatorily *trivial* – a trivial semantics of a logic program being one that contains the literals L and $\neg L$, for every atom L of the language of P [3]. This has been shown an important property for frameworks of knowledge and reasoning representation. This point is clarified by the next example.

Example 1.3. (adapted from [Ari02]) Consider the following program P.

$$\begin{aligned} a &\leftarrow r \\ q &\leftarrow \\ r &\leftarrow \\ \neg r &\leftarrow not\ \neg q. \end{aligned}$$

Both q and r should be true in any reasonable semantics of P. Now $\neg r$ is also true since $\neg q$ does not hold. This seems a natural interpretation of P, and hence P contains contradictory information regarding r. However the true value of q is not related to any contradictory information. The same cannot be said about a, which depends on r. Thus a paraconsistent semantics should be able to spot the true value of q as not supported on contradiction, and the true value of a as supported on contradiction, while r is itself contradictory (and thus supported on its own contradiction).

With special interest for this work is the paraconsistent *well-founded semantics with explicit negation* $WFSX_P$ [ADP95], which is a semantics of partial models for extended normal

[3]This characterization of a *paraconsistent* semantics is in line with Jakowski-da Costa's general definition of *paraconsistent theory*: given a logic with a negation "$\neg$", a theory T is paraconsistent iff it is inconsistent and non-trivial (see [dCBB95]).

logic programs. The $WFSX_P$ envisages default negation and explicit negation necessarily related through the *coherence principle* [PA92]: if $\neg L$ holds, then *not* L should also hold (similarly if L, then *not* $\neg L$). The $WFSX_P$ semantics of the program in example 1.3 is (see chapter 5)

$$WFM_P(P) = \{a, not\ a, not\ \neg a, r, not\ r, \neg r, not\ \neg r, q, not\ \neg q\}.$$

Notice that *default consistency* (i.e. L and *not* L cannot simultaneously be true) is not enforced by $WFSX_P$. This is a distinguish feature of this semantics, due to the adoption of the coherence principle, which permits detection of support on contradiction: when L and *not* L belong to a model, then L is supported on contradiction [ADP95]. The $WFSX_P$ reduces to the WFS on normal logic programs.

In chapter 5 we define the *paraconsistent minimal hypotheses* semantics MH_P, which combines the merits of the MH and of the $WFSX_P$. It is a semantics of *total paraconsistent models*, meaning that given an extended normal logic program P and a MH_P model M of P, model M contains no undefined literals. It endows $WFSX_P$ with choice mechanisms that allow reasoning by cases, in the style of MH. Let us illustrate this characterization of the MH_P with an example.

Example 1.4. Consider the following program P.

$$\begin{aligned} c &\leftarrow \\ a &\leftarrow not\ b \\ b &\leftarrow not\ a \\ \neg c &\leftarrow a \\ \neg c &\leftarrow b \\ r &\leftarrow c \end{aligned}$$

The $WFSX_P$ model of P is [4]:

$$WFM_P(P) = \langle \{c, r\}^+, \{a, b\}^u, \{\neg a, \neg b, \neg c, \neg r\}^- \rangle.$$

Notice that $\neg c \in WFM_P^-(P)$ due to the coherence principle. Now this model is not inconsistent, either with respect to default negation, or with respect to explicit negation [5]. The literal $r \in WFM_P^+(P)$ does not have support on contradiction, because the $WFSX_P$, being WFS based, retrieves a, b as undefined. Nevertheless it is argued in the literature that a reasonable semantics for normal logic programs should rely on the intersection of 2-valued models (see for instance [Dix96], section 3.5). As the subprogram $\{a \leftarrow not\ b,\ b \leftarrow not\ a\}$ is normal and depends on no other rule, a less skeptical semantics could retrieve either a or b as positive, and thus $\neg c$ would be positive and r would be supported on contradiction. It is the case that the MH_P semantics of P consists of the models

$$M_1 = \langle \{a, c, \neg c, r\}^+, \{\neg a, b, \neg b, c, \neg c, r, \neg r\}^- \rangle$$
$$M_2 = \langle \{b, c, \neg c, r\}^+, \{a, \neg a, \neg b, c, \neg c, r, \neg r\}^- \rangle,$$

[4] We represent the model with the alternative notation $M = \langle M^+, M^u, M^- \rangle$ and emphasize the valuations by using the corresponding label "$+, u, -$".

[5] We will see in chapter 5 that a contradiction with respect to default negation (L and *not* L) only occurs if a contradiction with respect to explicit negation also occurs with some literal L depends on.

both of them showing r as supported on contradiction, because r and *not* r belong to both models.

The main virtue of a paraconsistent semantics is that of preserving the information that is free from contradiction, in the presence of contradictory information, by allowing non trivial models. Notwithstanding, the existence of non-trivial models is of little use if some mechanism to separate this two types of information is not at hand. By detecting support on contradiction, the MH_P has incorporated such a mechanism for contradictions with respect to explicit negation. Meanwhile we may want to deal with broader inconsistency theories than the ones which consider only a single type of constraint, $\bot \leftarrow L, \neg L$. That is why in chapter 6 we consider two procedures to deal with inconsistent theories, with respect to constraint theories that contain more general constraints, say $\bot \leftarrow a, \textit{not}\ b$. In the presence of such more general constraints, the support on contradiction capability disposed by the MH_P semantics is not sufficient to detect all types of support on inconsistency that may now arise from the activation of integrity constraints. The procedures we expound there to deal with more general inconsistent theories, are the following. We consider a revision procedure that acts in the fashion of a declarative debugger [PDA93a], i.e., the revision proceeds by deleting rules of a theory or by revising the *CWA* valuation of certain literals, thus treating programs as malfunctioning devices needing repair. This type of revision approach is of a syntactic nature, inline with a family of approaches normally designated *belief base*, and thus different from other type of "semantic flavored" approaches normally designated *belief set* [DSTW12]. We consider also a procedure for detecting support on contradiction, where the contradictory literals are those that appear in the definition of the constraints whose bodies are verified by some model. Having such an inconsistency propagation detection procedure is of utmost importance, because: (1) it fulfills the very reason that underlies the existence of paraconsistent semantics, which is the possibility of extracting information from inconsistent models; (2) it may be used to get the sound information from an inconsistent theory, in cases where a revision process is not considered, or in cases where a revision is not capable of eliminating all inconsistencies in the resulting models.

In this introductory part of the thesis, we have already unveiled our proposals to accomplish the main goal of this work, which is to obtain a semantics that deals with inconsistent theories, and then face the revision process by resorting to an abductive mechanism in the style of the MH semantics. The fulfillment of this plan consists in the material presented in chapters 5 and 6.

Meanwhile some issues concerning the characterization of the MH_P with respect to the properties of *existence*, *relevance* and *cumulativity* were also tackled. Those are important properties from the computational point of view, as already referred. The main results of our approach on these matters, are stated in chapter 4. There we define a family of 2-valued conservative extensions of the SM semantics[6], the *affix stable model semantics* family ASM, whose elements are 2-valued semantics for normal logic programs. The purpose of establishing this family, is to present a general enough definition of 2-valued

[6]Meaning semantics that for each normal logic program retrieve all the stable models of the program, eventually together with some other models in order to ensure some or all of the properties of existence, relevance and cumulativity.

model conservative extensions of the stable model semantics, which eventually represents both a large and interesting number of semantics under this designation. In order to look for a characterization of some semantics pertaining to the ASM family, with respect to the properties of existence, relevance and cumulativity, we were led to the definition of two subfamilies $ASM^h \subset ASM$ and $ASM^m \subset ASM$. Semantics of the ASM^h family are minimal (positive) hypotheses generated abductive semantics, in the style of MH, whilst semantics of the ASM^m family are minimal models abductive semantics. We show that a semantics SEM of any of these two families is cumulative iff for any normal logic program P and any subset $S \subseteq \bigcap_{M \in SEM(P)} M^+$, we have $SEM(P) = SEM(P \cup S)$, that is, P and $P \cup S$ have exactly the same 2-valued models, where M^+ represents the positive literals in $M = \langle M^+, M^- \rangle$ and M^- comprises the atoms false by default. This is a very interesting result, in our opinion, since it represents a definition of cumulativity not in terms of sets of literals pertaining to all models of a program, but in terms of the models of the program themselves, not considering their intersection. We also show this property permits to spot cumulativity failure by means of counter-examples, even in some cases where the programs used to set the counter-examples do not explicitly exhibit a failure of cumulativity (see examples in section 4.3). By resorting to three semantics structural properties first defined in this work, *defectivity*, *excessiveness* and *irregularity*, the following relations are established for any semantics SEM of ASM^h or ASM^m families (see section 4.4):

1. Defectivity $\Leftrightarrow \neg$ Existence $\Leftrightarrow \neg$ Global to Local Relevance;
2. Defectivity $\Rightarrow \neg$ Cautious Monotony;
3. Excessiveness $\Rightarrow \neg$ Cut;
4. Irregularity $\Leftrightarrow \neg$ Local to Global Relevance,

where the properties *global to local relevance* and *local to global relevance* arise from splitting relevance into its two separate logic implications. All the results referred above may turn into an easier job both the defeating as well as the proof of the properties of *existence*, *relevance* and *cumulativity*, making it a matter of dealing with the structure of programs (i.e., the *layers* of normal logic programs [PP11]) and with the decomposition of models over that structure, thus avoiding in some cases proofs that may not be easy to obtain using a direct proof strategy. The structural approach taken to establish the profile of ASM^h and ASM^m families semantics, concerning the aforementioned properties, relies heavily on the concept of *layering* of normal logic programs [PP11], and is thus in line with a syntactic tackling of the semantics of normal logic programs.

The main results in this paper are enounced for the universe of finite ground logic programs, meaning programs without variables and where the rules, considered as sequences of symbols, have finite length.

The main original contributions of this work, to be detailed in the sequel, are:

1. An improvement in the definition of the MH semantics that reduces the set of assumable hypotheses of a program (see definition 3.16).

2. The concept of simple relevance of a semantics, that allows query answering in a brave reasoning fashion (see definition 3.18) .

3. The definition of cumulativity for the class $ASM^h \cup ASM^m$ of semantics, as a statement about sets of models instead of a statement about sets of atoms, which is advantageous for the study of this property (see theorem 4.9).

4. The structural concepts of defectivity, excessiveness and irregularity, on the basis of which some relations among the properties of existence, relevance and cumulativity were defined, with advantage for the study of theses properties (see definitions 4.18, 4.19, 4.20).

5. As an example of the relations among properties mentioned in the last item, we have shown that existence failure implies cautious monotony failure for the SM semantics. This result, to the best of our knowledge, had not yet been settled.

6. A detailed characterization of the MH semantics with respect to the properties of relevance and cumulativity.

7. The definition of the semantics *Cyan* that, to the best of our knowledge, is the first defined 2-valued fair conservative extension of the SM semantics that is rational, existential, relevant and cumulative.

8. A total models paraconsistent semantics MH_P, that is existential, simple relevant and has the capability of detecting support on contradiction (see definition 5.11).

9. A revision algorithm for inconsistent theories with respect to the MH_P semantics (see subsection 6.1.2).

10. An algorithm for computing the set of safe literals of the extended kernel of an inconsistent theory with respect to the MH_P semantics (see subsection 6.2.3).

We recommend reading the chapters in this thesis in the sequence they appear. However, if the reader is not interested in all the subjects, or is more keen on some topics rather than others, we provide alternative reading paths as shown below.

2-3-4-7	Study of the properties of existence, cumulativity and relevance in a subclass of the ASM family of 2-valued semantics.
2-3-5-6-7	Paraconsistent MH_P semantics, plus revision and inconsistency propagation detection methods (all sections in 2 and 3 are optional, but section 3.3)

Chapter 2

Structure of Noetherian Normal Logic Programs

We present some outlining terminology for the structure of normal logic programs adopted in this work.

We start by bring forward some usual terminology for normal logic programs, and some rule dependency notions, in sections 2.1 and 2.2. Then we introduce the notions of *layering* and *T-segment* of a normal logic program. Layering is essential to define the MH and MH_P semantics (chapters 3 and 5), whilst T-segment subtends all the structural properties definitions presented in chapter 4.

2.1 Language and Terminology of Logic Programs

A *normal logic program* defined over a language $\mathcal{L}$ is a set of rules, each one of the form

$$b_0 \leftarrow b_1, \cdots, b_m, not\ c_1, \cdots, not\ c_n \tag{2.1}$$

where m, n are integer non negative numbers and b_j, c_k are *atoms* of $\mathcal{L}$; b_i and $not\ c_k$ are generically designated *literals*, $not\ c_k$ being specifically designated *default literal* . The operator "," stands for the conjunctive connective, the operator *not* stands for negation by default and the operator "$\leftarrow$" stands for a *dependency* operator that establishes a dependence of b_0 on the conjunction on the right side of "$\leftarrow$". b_0 is the *head* of the rule and $b_1, \cdots, b_m, not\ c_1, \cdots, not\ c_n$ is the *body* of the rule. A rule is named a *fact* if $m = n = 0$. A program is *definite* if it has no default literals. A literal (program) is *ground* if it does not contain variables. The set of all ground atoms of a normal logic program is called *Herbrand base* of P, $\mathcal{H}_P$. The set of all ground terms of P is called *Herbrand universe* of P, $\mathcal{U}_P$. If r is a rule of P, by its *ground instance* we mean any ground rule obtained from

r by substituting ground terms from $\mathcal{U}_P$ for all variables. A program is *finite* if it has a finite number of rules [1]. Given a program P, program Q is a *subprogram* of P if $Q \subseteq P$. Although the main results in this thesis consider only finite ground logic programs, the notions defined in this chapter envisage the class of all ground normal logic programs with a countable (possibly infinite) set of rules.

For ease of exposition we henceforth use the following abbreviations: $Atoms(E)$, is the set of all *atoms* that appear in the ground structure E, where E can be a rule, a set of rules, a set of logic expressions, etc; $Bodies(E)$, is the set of all bodies that appear in the set of ground rules E; if E is unitary, we may use "*Body*" instead of "*Bodies*"; depending on the context, *Body* may refer to the conjunction of literals that forms the body of a rule (as opposed to the set formed by those literals); $Heads(E)$, is the set of all atoms that appear in the heads of the set of rules E; if E is unitary, we may use "*Head*" instead of "*Heads*"; $Facts(E)$, is the set of all facts that appear in the set of rules E; $Loops(E)$, is the set of all rules that pertain to a loop contained in the set of rules E (see definition 2.6 for loop). We may compound some of these abbreviations, as for instance $Atoms(Loops(P))$ or $Atoms(Bodies(Loops(P)))$ whose meaning is immediate.

2.2 Rule Dependencies

We establish some terminology concerning the dependencies among the elements of a normal logic program (atoms and rules) triggered by the dependency operator "$\leftarrow$".

Definition 2.1. Complete rule graph. (adapted from [Pin11]) Let P be a normal logic program. The *complete rule graph* of P, denoted by $CRG(P)$, is the directed graph whose vertices are the rules of P. Two vertices representing rules r and s define an arc from r to s iff $Head(r) \in Atoms(Body(s))$.

Definition 2.2. Rule depending on a rule. (adapted from [Pin11]) Let r, s be two rules of a normal logic program P. We say that s *depends* on r iff there is a directed path in $CRG(\mathcal{P})$ from r to s.

Definition 2.3. Strongly connected components. (adapted from [Tar72, Pin11]) Let P be a normal logic program. We dub *strongly connected component*, *SCC*, of P, any maximal subset Q of P, such that for any two rules of Q, say r, s, rule r depends on rule s and rule s depends on rule r.

Definition 2.4. Atom depending on a rule. (adapted from [Pin11]) We say that an atom b depends on a rule r in a normal logic program P iff there is a rule s, where $Head(s) = b$, such that s depends on r. In particular b depends on s.

Definition 2.5. Subprogram relevant to an atom. (adapted from [Dix95b, Pin11]) Let P be a normal logic program. We say that a rule $r \in P$ is *relevant* to an atom $a \in \mathcal{H}_P$ iff a depends on r. The set of all rules of P relevant to a is represented by $Rel_P(a)$, and is named *subprogram (of P) relevant* to a.

[1]In this work, if nothing else is said, by logic program we mean a finite set of ground rules.

Definition 2.6. Loop. (adapted from [Cos95]) We say that a set of ground rules P forms a *loop* iff P is of the form

$$\begin{aligned}
&h_1 \leftarrow l_2, B_1\\
&h_2 \leftarrow l_3, B_2\\
&\cdots\\
&h_k \leftarrow l_{k+1}, B_k\\
&\cdots\\
&h_n \leftarrow l_1, B_n
\end{aligned}$$

where $l_i = h_i$, or $l_i = not\ h_i$, and each B_i (may not exist) stands for the conjunction of a finite number of literals. We say that each rule $h_i \leftarrow l_{i+1}, B_i$ is *involved* in the loop *through the literal* l_{i+1} or *through the atom* involved in the literal l_{i+1}, where $i+1$ is replaced by 1 if $i = n$. We say that the loop is an *even loop over negation* (resp. *odd loop over negation*), if the set of literals $\{l_1, l_2, \cdots, l_{n-1}\}$ contains an even (resp. odd) number of default literals. In a SCC every rule is involved in at least one loop.

Definition 2.7. Module of a normal logic program. (adapted from [Pin11]) Let P be a normal logic program. Any SCC in P is called a *module* of P. A rule not pertaining to any loop of P is also called a *module* of P. Nothing more is a module of P. We denote the class of all modules of P by $Modules(P)$.

Definition 2.8. Modules graph of a normal logic program. (adapted from [Pin11]) Let $CRG(P)$ be the complete rule graph of a normal logic program P. We call *modules graph* of P to the graph $\mathcal{MG}(P) = (Modules(P), Arcs)$, where $Arcs = \{(x, y) : x, y \in Modules(P)\}$ such that there is a directed path in the graph $CRG(P)$ from a rule of the module x to a rule of module y. The modules graph of a normal logic program P is unique, as per this definition.

2.3 Noetherian Normal Logic Programs

We define the *rule layering* of a normal logic program, which labels every rule in the program with a natural number. We consider only *noetherian* normal logic programs, which are programs that contain no infinite dependence descending chains of rules.

Definition 2.9. Infinite descending dependence chain. We say that an infinite set S of ground rules forms an *infinite descending dependence chain* iff the set of rules is of the form:

$$\begin{aligned}
&h_1 \leftarrow l_2, B_1\\
&h_2 \leftarrow l_3, B_2\\
&\cdots\\
&h_n \leftarrow l_{n+1}, B_n\\
&\cdots
\end{aligned}$$

where $(j \neq k) \Rightarrow (h_j \neq h_k)$, $j, k \in \mathbb{N}$ and each l_i is either h_i or $not\ h_i$.

We are interested in characterize the structure of any ground normal logic program with no infinite descending chains.

Definition 2.10. Noetherian logic program. We say that a logic program P is *noetherian*[2] iff it does not contain infinite descending dependence chains.

The following proposition is a characterization of the *modules graph* of a noetherian normal logic program.

Proposition 2.1. The modules graph $\mathcal{MG}(P) = (Modules(P), Arcs)$ of a noetherian normal logic program P, contains at least one vertex with zero indegree.

Proof. Suppose that $\mathcal{MG}(P)$ contains no vertex with zero *indegree*. The argument below shows that this hypothesis leads to a contradition.

1. Chose any module of $Modules(P)$ and name it N_1.
2. As $indegree(N_1) \neq 0$, there is a rule $r_1 \in N_1$ that depends on a rule r_2 of another module. Let N_2 be this other module.
3. As $indegree(N_2) \neq 0$, rule r_2 must depend on another rule $r_3 \notin N_2$, where $r_3 \notin N_1$ otherwise N_1 and N_2 would not be different modules. Let N_3 be the module containing r_3.
4. Keeping this reasoning going on, we obtain a descending chain of rules, $r_1, r_2, r_3, \cdots$, that is infinite unless a module with zero indegree exists, which must be the case since P is noetherian.

$\square$

We now set forth a constructive definition of *rule layering* [Pin11] of a noetherian normal logic program P.

Definition 2.11. Rule layering. (adapted from [Pin11]) Let $\mathcal{MG}(P)$ be the modules graph of a noetherian normal logic program P and $\mathbb{N}$ the set of natural numbers. We define the *rule layering* (or just *layering*, for simplicity) of P as the labeling of rules corresponding to the outcome of the total *layer function*, $\mathcal{MG}(P) \mapsto \mathbb{N}$, inductively defined as follows:

Step 1 Let $\mathcal{MG}_1(P) = \mathcal{MG}(P)$. Label with 1 all the rules in the zero indegree vertices of $\mathcal{MG}_1(P)$. Erase from $\mathcal{MG}_1(P)$ all the zero indegree vertices and the arcs emerging from them, and name $\mathcal{MG}_2(P)$ the resulting graph.

Step k Label with k all the rules in the zero indegree vertices of $\mathcal{MG}_k(P)$. Erase from $\mathcal{MG}_k(P)$ all the zero indegree vertices and the arcs emerging from them, and name $\mathcal{MG}_{k+1}(P)$ the resulting graph.

Every integer number T in the image of the layer function defines a *layer* of P, meaning the set of rules of P labeled with number T.

The process of labeling in the above definition finishes if P is finite. By then:

[2]The designation "noetherian" is here adopted after the paper [Dix95b]

- Every rule of the normal logic program is labeled with an integer positive number;
- The rules belonging to the same module have the same label;
- The highest label involved in the process corresponds to the number of layers of P.

This concept of rule layering coincides with the notion of *least rule layering* presented in [Pin11].

Proposition 2.2. ***Existence and unicity of layering.*** (adapted from [Pin11]) Every normal logic program has a unique layering.

Proof. Immediate as per definition 2.11. □

Through out this thesis we adopt the following conventions:

- Labels are denoted either by capital letters or else by natural numbers;
- Expressions such as "layer T" of a program, mean the set of rules with the label T in the program;
- Expressions such as "a set of rules is in layer T", mean that all the rules of the set are labeled with the label T;
- To say that a set of rules is above (resp. below) the layer T, means that all the rules of the set have labels greater (resp. lesser) than T.

The following representations [Pin11] are adopted in this thesis:

- $P^{\leq T}$ (resp. $P^{\geq T}$) is the subprogram of P consisting of the union of the rules in layer T and layers below (resp. above) T.
- $P^{<T}$ (resp. $P^{>T}$) is the subprogram of P consisting of the union of the rules in layers below (resp. above) the layer T.
- P^{T} is the subprogram of P consisting of the rules in layer T.

Example 2.1. ([Pin11]) The following schema represents a program with the rules already separated into layers by the dashed lines. The labels of the layers are the integer numbers T before the dashed lines, and each one of them refers to the sets of rules immediately above, if $T = 1$, or to the set of rules between the dashed lines $T - 1$ and T, if

$T > 1$.

$$
\begin{array}{l}
x \leftarrow not\ x \\
e \leftarrow e \\
f \leftarrow \\
1 - - - - - - \\
b \leftarrow not\ x \\
y \leftarrow not\ x \\
z \leftarrow f \\
2 - - - - - - \\
b \leftarrow not\ b \\
d \leftarrow not\ c \\
c \leftarrow not\ d, not\ y, not\ a \\
3 - - - - - -
\end{array}
$$

In the sequel, it will be useful the concept of *T-segment* defined below.

Definition 2.12. T-segment of a normal logic program. Let P be a normal logic program and T a layer of P. We say that $P^{\leq T}$ is the *T-segment* of P iff $Atoms(P^{\leq T}) \cap Heads(P^{>T}) = \emptyset$.

Example 2.2. In the program of example 2.1 the 1-segment of P, $P^{\leq 1}$, is the set of rules above dashed line 1. Notice that the set of rules above dashed line 2 is not a segment of the program, since $Atoms(P^{\leq 2}) \cap Heads(P^{>2}) = \{b\} \neq \emptyset$. Hence the program does not have a 2-segment.

Chapter 3

Semantics for Normal Logic Programs

We define some semantic concepts needed ahead in this work, and make a brief presentation of a number of 2-valued semantics for normal logic programs in order to provide background for the chapters that follow.

The semantics of a logic program describes the intended meaning of the program, by specifying its logical consequences. For logic programs two manners to specify a semantics are relevant: *procedural semantics* and *declarative semantics*. A procedural semantics provides an implementation-independent proof procedure from which logical consequences of programs are derived. A declarative semantics is specified by a model theory. Model theories define interpretations with respect to which a logic program is valid, meaning that all the rules of the program are satisfied. This is fulfilled by resorting to the *Herbrand universe* and *Herbrand interpretations* of a logic program. In this work we follow closely the model-theoretical approach to define normal logic programs semantics. Every semantics SEM we define/refer consists, for each normal logic program, in a subset of the set of all classical models of the program. For a normal logic program P, the set $SEM(P)$ is the set of all the SEM models of P, designated by *semantics of P with respect to SEM* or by *SEM semantics of P*.[1]

The remaining of this chapter goes as follows. In section 3.1 we define the notions of interpretation, model and T-segment decomposition of a model. In section 3.2 we present a brief overview of some normal logic programs semantics (several overviews of the subject can be found in literature, e.g., [AB94, Dix96]). Finally in section 3.3 we define in detail the MH semantics.

[1]This first paragraph follows closely some parts of the works [LMR92, Alf93].

3.1 Interpretations and Models

We next define 2-valued and 3-valued Herbrand interpretations of normal logic programs.

Definition 3.1. Interpretation. (adapted from [DP98]) An interpretation I of a normal logic program P is a 3-tuple of the form

$$I = \langle I^+, I^u, I^- \rangle, \tag{3.1}$$

where $\mathcal{H}_P = I^+ \cup I^u \cup I^-$ and I^+, I^u, I^- represent the sets of atoms of the language of P that are respectively positive, undefined and default false in I. If $I^u = \emptyset$ (resp. $I^u \neq \emptyset$), then the interpretation is said to be a *total* (resp. *partial*) interpretation or a 2-*valued* (resp. 3-*valued*) interpretation.

For convenience, we may sometimes represent a 2-valued interpretation I by $I = I^+ \cup$ *not* I^-, or by $I = \langle I^+, I^- \rangle$, where *not* I^- is the set of all the default negations of atoms of I^-. In some cases we may represent a 2-valued interpretation I by I^+, being understood that $I^- = \mathcal{H}_P \setminus I^+$. We may also represent a 3-valued interpretation I by $I = I^+ \cup$ *not* I^-, being understood that $I^u = \mathcal{H}_P \setminus (I^+ \cup I^-)$.

Definition 3.2. Satisfaction. (adapted from [DP98]) Let I be an interpretation of a normal logic program P. We say that,

1. I falsifies a literal b (resp. *not* b) iff $b \in I^-$ (resp. $b \in I^+$); I falsifies the body of a rule $r \in P$ iff I falsifies at least one literal of $Body(r)$;

2. I verifies a literal b (resp. *not* b) iff $b \in I^+$ (resp. $b \in I^-$); I verifies the body of a rule $r \in P$ iff I verifies all literals of $Body(r)$;

3. I makes undefined a literal b (*not* b) iff $b \in I^u$; I makes undefined the body of a rule $r \in P$ iff I makes undefined a nonempty set of literals of $Body(r)$ and verifies all the remaining ones;

4. I falsifies a rule r iff I falsifies $Head(r)$ and does not falsify $Body(r)$, or if I makes undefined $Head(r)$ and verifies $Body(r)$; I verifies a rule r iff I does not falsify r.

We denote $I \models l$ (resp. $I \models r$) the satisfaction of a literal l (resp. a rule r) by an interpretation I.

Definition 3.3. Model. [DP98] An interpretation I is a *model* of a normal logic program P iff all rules in P are satisfied by I.

If b is an atom of the language of P and M is a model of P with respect to some semantics SEM, then we may state, for simplicity, that $b \in M^+$ (resp. $b \in M^u$, $b \in M^-$) by writing $b = +$ (resp. $b = u$, $b = -$). A semantics SEM is 2-valued iff all SEM models it retrieves are 2-valued models. A semantics SEM is 3-valued iff there is at least a normal logic program P for which $SEM(P)$, the set of SEM models of P, contains at least one 3-valued model.

We next present the structural concept of T-*segment decomposition* of a model, concerning 2-valued semantics, which will be used in the next chapter to characterize families of 2-valued conservative extensions of the SM semantics on the formal properties of existence, relevance and cumulativity.

Definition 3.4. ***T*-segment decomposition of a model.** Let P be a normal logic program, $P^{\leq T}$ a segment of P, SEM a 2-valued semantics and $M \in SEM(P)$. We call *T-segment decomposition* of the model M, to the 3-valued interpretation $M_{\leq T} = M^{+}_{\leq T} \cup not\ M^{-}_{\leq T}$, where $M^{+}_{\leq T} = M^{+} \cap Atoms(P^{\leq T})$ and $M^{-}_{\leq T} = M^{-} \cap Atoms(P^{\leq T})$.

The next operator shall be used in the sequel.

Definition 3.5. **$\triangle$ operator.** Given a normal logic program Q, we denote by $\triangle Q$ the 3-valued interpretation that can be read from Q in the following way: $b \in (\triangle Q)^{+}$ iff $(b \leftarrow) \in Q$; $b \in (\triangle Q)^{u}$ iff $(b \leftarrow) \notin Q$ and there is a rule r in Q such that $Head(r) = b$; $b \in (\triangle Q)^{-}$ iff b has no rule in Q.

The following condition is obeyed by all the semantics treated in this thesis. It is an extension of a property named C_1 in [PAA92] – we adopt the designation C_1 for the variant of this property here presented.

Definition 3.6. **$\mathbf{C_1}$ property.** We say that a semantics SEM for normal logic programs obeys the property C_1 iff for any normal logic program P, any model $M \in SEM(P)$ and any rule $r \in P$, the following conditions are verified: (1) if M satisfies $Body(r)$ then $Head(r) \in M^{+}$ and rule r is satisfied by M; (2) if M falsifies $Body(r)$, then rule r is satisfied by M.

The following two orderings will be used through out the sequel.

Definition 3.7. **Classical truth ordering.**[2] The interpretations $I_1 = I_1^{+} \cup not\ I_1^{-}$ and $I_2 = I_2^{+} \cup not\ I_2^{-}$, satisfy the *classical truth ordering relation* $I_1 \leq_t I_2$, iff $I_1^{+} \subseteq I_2^{+}$ and $I_2^{-} \subseteq I_1^{-}$.

Definition 3.8. **Knowledge ordering.**[3] The interpretations $I_1 = I_1^{+} \cup not\ I_1^{-}$ and $I_2 = I_2^{+} \cup not\ I_2^{-}$, satisfy the *knowledge ordering* relation $I_1 \leq_K I_2$, iff $I_1^{+} \subseteq I_2^{+}$ and $I_1^{-} \subseteq I_2^{-}$.

3.2 Minimal Models Semantics

The motivation behind *minimal models* semantics, is based on the idea that one should minimize positive information in the models as much as possible [AP96], limiting it to facts explicitly implied by the program, and making everything else false. This may be taken as an application of the *Ockham's razor* principle to logic programs semantics [Pin11]. The minimal models semantics briefly presented in this section are the *least model semantics*, the *stable model semantics* and the *well-founded semantics.*

3.2.1 Least Model Semantics

The *least model semantics* [vEK76] is a semantics for *definite* logic programs. Given a definite normal logic program P, the least model of P is the least fix-point of the T_P operator defined below.

[2]Given the logic values f (false), u (undefined) and t (true), their *truth ordering* is defined by $f \leq_t u \leq_t t$ [AP96].

[3]Given the logic values f (false), u (undefined) and t (true), their *knowledge ordering* is defined by $u \leq_t f$ and $u \leq_t t$ [AP96].

Definition 3.9. $\mathbf{T_p}$ operator. (adapted from [Prz90a]) Let P be a definite ground logic program and I a 2-valued interpretation. Then

$$T_P(I) = \{a : a \leftarrow a_1, \cdots, a_n \in P \text{ and } \{a_1, \cdots, a_n\} \subseteq I\}. \tag{3.2}$$

Definition 3.10. Least model semantics. (adapted from [Prz90a]) The *least* 2-valued model M_P of a definite ground logic program P is equal to $T_P^{\uparrow\omega}(\emptyset)$.

3.2.2 Stable Model Semantics

The stable models of a normal logic program may be computed as fixpoints of the Gelfond-Lifschitz operator Γ defined below. This operator takes as arguments a normal logic program P and a 2-valued interpretation of it I, and retrieves the 2-valued interpretation $\Gamma_P(I)$.

Definition 3.11. $\mathbf{\Gamma}$ operator. (adapted from [Prz90a]) Let P be a normal logic program and I a 2-valued interpretation. The Gelfond-Lifschitz *transformation of P modulo I*, is the program $\frac{P}{I}$ obtained from P by performing the following transformations:

1. Remove from P all rules which contain a default literal *not b*, for every literal $b \in \mathcal{H}_P$ such that $b \in I^+$;
2. Remove from the remaining rules all default literals.

Since the resulting program $\frac{P}{I}$ is definite, it has a unique least 2-valued model M. We define $\Gamma_P(I) = M$.

Definition 3.12. Stable model semantics, *SM*. (adapted from [Prz90a]) A 2-valued interpretation I is a *stable model* of a normal logic program P iff $\Gamma_P(I) = I$.

3.2.3 Well-Founded Semantics

In [BDFZ01] the authors propose a reduction system comprising the following five operations, each of which transforms normal logic programs into normal logic programs while keeping invariant the *well-founded model* [Gel93] of the programs: *positive reduction*, $\mapsto_P$, *negative reduction*, $\mapsto_N$, *success*, $\mapsto_S$, *failure*, $\mapsto_F$, and *loop detection*, $\mapsto_L$ (see definitions in appendix A). We here represent this reduction system by $\mapsto_{WFS}:=\mapsto_P \cup \mapsto_N \cup \mapsto_S \cup \mapsto_F \cup \mapsto_L$. Given a normal logic program P, the transformation $P \mapsto^*_{WFS} \widehat{P}$ (where $\mapsto^*_{WFS}$ means the non deterministic performing of operations of the system until the resulting program becomes invariant) is such that $WFM(\widehat{P}) = WFM(P)$. Program $\widehat{P}$ will be here called the *WFS remainder* of P, or *program remainder* of P, or simply *remainder* of P (these latter two designations are used in [BDFZ01]). The system $\mapsto_{WFS}$ is both *terminating* and *confluent*, meaning that for any finite ground normal logic program the number of operations needed to reach $\widehat{P}$ is finite, and the order in which the operations are performed is irrelevant.

Definition 3.13. Well-founded semantics, *WFS*. (adapted from [Gel93, BDFZ01]) The *well-founded model* of a normal logic program P, $WFM(P)$, is the interpretation $\triangle\widehat{P}$, where $P \mapsto^*_{WFS} \widehat{P}$.

3.3 The Minimal Hypotheses Semantics, MH

In this section we state the role of abduction in obtaining a semantics of total models, and after we define with technical detail the MH semantics.

3.3.1 2-Valued Semantics through Abduction

Abductive logic programming is an extension of logic programming to perform abductive reasoning [DK02]. The general definition of the abductive task, in the context of logic programming, is as follows.

Definition 3.14. Abductive task, abductive semantics, hypotheses. (adapted from [DK02]) Given a normal logic program P and a query Q, the *abductive task* can be characterized as the problem of finding a set of literals Ω (*abductive explanation* for Q or *hypotheses* for Q), from the language of P, such that $P \cup \Omega \models Q$, where the relation "$\models$" is in accordance with a certain set of criteria for rule satisfaction. The semantics obtained as a result of this task is an *abductive semantics*.

In what concerns the SM semantics, the consideration of positive hypotheses involving atoms from default literals, allows to obtain the single stable model of any logic program P whenever $WFM^u(P) = \emptyset$ (the single stable model of P coincides with its well-founded model). In this case, the abductive task takes $\Omega = \emptyset$, and $P \models Q$, meaning that $WFM(P)$ satisfies Q. Moreover, one may also obtain the stable models that result from solving even loops over negation, by resorting to sets of positive hypotheses Ω whose elements are atoms that appear in default literals involved in such loops [PP11]. This approach can be extended to obtain a semantics for normal logic programs that otherwise lack stable models, as shown in the next example.

Example 3.1. In order for a program like P

$$\begin{aligned} a &\leftarrow not\ b \\ b &\leftarrow not\ c \\ c &\leftarrow not\ a \end{aligned}$$

to have a 2-valued model M, a subset of $\{a, b, c\}$ must be a part of M^+. This can be achieved by considering *abductive extensions* [DK02] $P \cup \Omega$ of the program P, where Ω is a subset of $\{a, b, c\}$, such that $P \cup \Omega$ has a stable model that is equal to its well-founded model [PP11]. For example, if we consider the explanation $\Omega = \{a\}$, the stable model obtained for $P \cup \Omega$, which coincides with the $WFM(P \cup \Omega)$, is $\{a, b\}$. In case we take, for instance, the explanation $\Omega = \{b\}$ (resp. $\Omega = \{c\}$), we get the stable model $WFM(P \cup \{b\}) = \{b, c\}$ (resp. $WFM(P \cup \{c\}) = \{c, a\}$).

Abductive semantics allow us to envisage loops in normal logic programs as semantic choice devices. This is one of the main features of the *minimal hypotheses* semantics, MH.

3.3.2 MH Models

To compute the MH semantics of a normal logic program, we need the variant $\mapsto_{LWFS}$ of the reduction system $\mapsto_{WFS}$, which results from $\mapsto_{WFS}$ by substituting the negative reduction operation $\mapsto_N$, by the *layered negative reduction* operation, $\mapsto_{LN}$.

Definition 3.15. Layered negative reduction. (adapted from [PP11]) Let P_1 and P_2 be two ground normal logic programs. Program P_2 results from P_1 by *layered negative reduction*, $P_1 \mapsto_{LN} P_2$, iff there is a rule $r \in P_1$ and a default literal $not\ b \in Body(r)$ such that $b \in Facts(P_1)$, r is not in loop through b, and $P_2 = P_1 \setminus \{r\}$.

Given a normal logic program P, the transformation $P \mapsto^*_{LWFS} \mathring{P}$ (where $\mapsto^*_{LWFS}$ means the non deterministic performing of operations of the system $\mapsto_{LWFS}$ until the resulting program becomes invariant) is such that $\triangle\mathring{P} = LWFS(P)$, where $LWFS(P)$ stands for the *layered well-founded model* of P [Pin11] – the transformed program $\mathring{P}$ is called *LWFS remainder* or *layered remainder* of program P (the latter designation is used in [PP11]).

Theorem 3.1. (adapted from [PP11]) The system $\mapsto_{LWFS}$ is terminating and confluent when applied to any finite ground normal logic program.

Proof. An immediate consequence of lemma 5.17. □

Example 3.2. The layered remainder of program P below (left column) is the program $\mathring{P}$ (right column) (the literals and rules eliminated in the computation of $\mathring{P}$ are striped out):

$b \leftarrow h$	$b \leftarrow h$
$h \leftarrow not\ p, b$	$h \leftarrow not\ p,$ ~~b~~
$p \leftarrow not\ b$	$p \leftarrow not\ b$
$a \leftarrow not\ c, b$	$a \leftarrow$ ~~$not\ c, b$~~
$d \leftarrow not\ b$	~~$d \leftarrow not\ b$~~
$b \leftarrow$	$b \leftarrow$

Notice that rule $d \leftarrow not\ b$ in P is eliminated in $\mathring{P}$ by layered negative reduction, the body of rule $a \leftarrow not\ c, b$ became empty by success (which eliminates b) and positive reduction (which eliminates $not\ c$), and b is eliminated from the body of rule $h \leftarrow not\ p, b$ by success. The layered well-founded model of P is thus $LWFM(P) = \triangle\mathring{P} = \langle \{a, b\}^+, \{h, p\}^u, \{c, d\}^- \rangle$.

MH being an abductive semantics, we shall now define the *assumable hypotheses set* and the *minimal hypotheses* model of a normal logic program.

Definition 3.16. Assumable hypotheses set of a program. (adapted from [PP11]) Let P be a finite normal logic program. We write $Hyps(P)$ to denote the *assumable hypotheses set* [4] of P: those atoms that appear default negated in the bodies of rules of $\mathring{P}$ and which are not facts in $\mathring{P}$.

All the assumable hypotheses of a normal logic program P belong to $LWFM^u(P) = (\triangle\mathring{P})^u$, as per the definition above. The purpose of computing $\mathring{P}$, is thus to find the set of assumable hypotheses of P, which are then used to compute the minimal hypotheses models of the program.

[4] This represents a slight variation on the original definition in [PP11] (where all the atoms that appear default negated in the bodies of rules of $\mathring{P}$ are taken as assumable hypotheses). This new definition reduces the set of assumable hypotheses, without changing the MH models obtained.

Definition 3.17. Minimal hypotheses model. (adapted from [PP11]) Let P be a finite normal logic program. Let $Hyps(P)$ be the assumable hypotheses set of P, and H a subset of $Hyps(P)$. A 2-valued interpretation M of P is a *minimal hypotheses* model of P iff $M = WFM(P \cup H)$, being $WFM^u(P \cup H) = \emptyset$, where $H = \emptyset$ or H is a nonempty set that is minimal with respect to set inclusion (set inclusion minimality concerns only nonempty Hs [5]). I.e., the hypotheses set H is minimal but sufficient to determine (via the well-founded model) the truth-value of all literals in the program.

The following result states the *existence* property of the MH semantics, i.e., every normal logic program has a MH semantics.

Theorem 3.2. (adapted from [PP11]) Every normal logic program has at least one MH model.

Proof. Given a normal logic program P, either some minimal model is a minimal hypotheses model, or else no minimal model of P is a minimal hypotheses model. This last case means there must exist a set of minimal hypotheses models that out rules the minimal models from the set of all the MH models of P. Whatever the case may be, a minimal hypotheses model always exists.[6] □

Every stable model of a normal logic program is also a minimal hypotheses model of the program. This justifies the catering for whole models with empty hypotheses sets $H = \emptyset$, which are stable models of programs whose layered remainders are stratified programs. The reason hypotheses minimization does not contemplate empty hypotheses sets models, is to allow loops to be taken as choice devices. For instance, program P in example 3.2 has the assumable hypotheses set $Hyps(P) = \{p\}$, since *not p* appears in $\mathring{P}$ and p is not a fact of $\mathring{P}$ [7]. The MH models of P are

$$\begin{array}{lll} \{a, b, not\ c, h, not\ p\} & \text{with hypotheses set} & \emptyset \\ \{a, b, not\ c, not\ h, p\} & \text{with hypotheses set} & \{p\}. \end{array}$$

If the empty hypotheses set were allowed in the hypotheses minimization, the non-empty hypotheses set model would be discarded and we would be left with just the stable model $\{a, b, not\ c, h, not\ p\}$.

Example 3.3. (adapted from [PP11]) Consider the following variation P of the *vacation problem*: five friends are planning a joint vacation. First friend says "If we don't go to place b, then we should go to place a", which corresponds to rule $a \leftarrow not\ b$; the same rationale for the remaining rules.

$$\begin{aligned} &a \leftarrow not\ b \\ &b \leftarrow not\ a, not\ c \\ &c \leftarrow not\ d \\ &d \leftarrow not\ e, not\ a \\ &e \leftarrow not\ a, not\ c \end{aligned}$$

[5] This point is clarified in the sequel.

[6] See example 3.3 to verify that a MH model may not be minimal.

[7] Notice that although *not b* appears in $\mathring{P}$, b is not an assumable hypothesis of P since it is a fact of $\mathring{P}$.

We have $P = \mathring{P}$. The hypotheses set of P is $Hyps(P) = \{a, b, c, d, e\}$. The MH models of P are

$$\begin{array}{lll} \{a, not\ b, c, not\ d, not\ e\} & \text{with hypotheses set} & \{a\} \\ \{not\ a, b, not\ c, d, e\} & \text{with hypotheses set} & \{b, d\} \\ \{a, not\ b, c, not\ d, e\} & \text{with hypotheses set} & \{e\}. \end{array}$$

Notice that there are no MH models with hypotheses sets $H = \{b\}$ or $H = \{d\}$, since in these cases $WFM^u(P \cup H) \neq \emptyset$. Notice also that the model $\{a, not\ b, c, not\ d, e\}$ is not minimal.

The MH solution of the vacation problem has the following rationale: rule $a \leftarrow not\ b$, for example, states that the first friend prefers place b to place a, because a is suggested in case b fails; each model of MH tries to satisfy the first options of the friends, by considering them as hypotheses. The answer set solution to this type of problem, when it exists, retrieves models that satisfy all the friends demands (rules) with the smallest (with respect to set inclusion) possible number of places to visit (due to the minimality of models). The example above shows this type of problem is not generally solvable by resorting to answer sets semantics [Lif08], if we stick to the set of rules of P, since models may not be minimal (e.g. $\{a, not\ b, c,\ not\ d, e\}$ above). Should there be a transformation on normal logic programs, let it be $\mapsto_Y$, such that $P \mapsto_Y P^*$, where the MH models of P could be extracted from the stable models of P^*, then P^* would have a different set of rules and/or a different language, with respect to P, which means that this type of problem is specified in a more elegant way if the solution is to be obtained via the MH semantics. Yet, it is an open problem whether such a transformation exists.

3.3.3 Formal Properties of MH

A detailed characterization of the MH semantics with respect to the formal properties of relevance and cumulativity is found in appendix B.3. Meanwhile the MH enjoys the existence property, by theorem 3.2, and the *simple relevance* property, as shown below.

Definition 3.18. Simple relevance. We say that a semantics SEM has the property of *simple relevance* iff for any normal logic program P, whenever there is a SEM model M_l of $Rel_P(l)$ such that $l \in M_l$, there is also a SEM model M of P such that $l \in M$.

Proposition 3.3. *MH semantics is simply relevant.*

Proof. Let $M_l \in MH(Rel_P(l))$, with hypotheses H_l, and $l \in M_l$. Let $P^* = P \cup M_l^+$ and $M \in MH(P^*)$ with hypotheses H_M, $H_M \cap H_l = \emptyset$ – we can see such a model must exist, by making for example $H_M = \emptyset$, if $WFM^u(P^*) = \emptyset$, or else H_M being chosen among the atoms that appear default negated in $\widehat{P^*}$, if $WFM^u(P^*) \neq \emptyset$. Then $l \in M$. Now it is the case that $M \in MH(P)$ because $H = H_M \cup H_l$ is a minimal hypotheses set for M by the following argument:

1. Either $H_M \cup H_l = \emptyset$ and then $M = WFM(P)$ is a MH model of P, where $WFM^u(P) = \emptyset$, as per definition 3.17, or else,
2. If $b \in H_l$, then $WFM^u(P \cup (H \setminus \{b\})) \neq \emptyset$ since $Rel_P(l)$ is not solved – notice that $Heads(P \setminus Rel_P(l)) \cap Atoms(Rel_P(l)) = \emptyset$, and

3. If $b \in H_M$, then $WFM^u(P \cup (H \setminus \{b\})) \neq \emptyset$ since P^* is not solved;
4. As $l \in M$ the proposition is proved.

$\square$

Chapter 4

Conservative 2-Valued Extensions of the Stable Model Semantics

We study strong and weak properties of some conservative 2-valued extensions of the stable model semantics, under a structural point of view.

The SM semantics is generally accepted by the scientific community working on logic programs semantics as the *de facto* standard 2-valued semantics. Nevertheless there are some advantageous properties the SM semantics lacks such as (1) model existence for every normal logic program, (2) relevance and (3) cumulativity [PP11]. It is known that all logic programs that lack stable models contain odd loops over negation [Cos95]. It seems thus reasonable to cater for models that afford odd loops, in order to accomplish a 2-valued semantics for every normal logic program, whilst maintaining all other stable models. This goal can be achieved by considering 2-valued conservative model extensions of the SM semantics [PP11], where *conservative* means that for each normal logic program all its stable models are obtained, possibly together with additional models. In this paper we define a family of 2-valued conservative extensions of the SM semantics, the *affix stable model semantics* family, ASM. The purpose of establishing this family is to present a general enough definition of 2-valued model conservative extensions of the stable model semantics, which eventually represents both a large and interesting set of semantics under this designation. The family ASM is presented as a subset of a broader class, the *fair* semantics family, which encompasses for sure a number of 2-valued semantics, so very little demanding are the conditions in its definition. We define two subfamilies of semantics, $ASM^h \subsetneq ASM$ and $ASM^m \subsetneq ASM$, and characterize them with respect to the properties of relevance and cumulativity.

The structural approach taken in this chapter to establish the profile of semantics from the families ASM^h and ASM^m, concerning the aforementioned properties, relies heavily on the concept of *layering* of normal logic programs [PP09], and is thus inline with a syntactic tackling of the semantics of normal logic programs.

The remainder of this chapter proceeds as follows. In section 4.1 families ASM, ASM^h and ASM^m are defined, as well as some semantics pertaining to them. In section 4.2 we define the formal properties we are dealing with for characterizing the families ASM^h and ASM^m. In section 4.3 we study the property of cumulativity for the families ASM^h and ASM^m, while in section 4.4 some results involving defectivity, excessiveness and irregularity are set forth. Section 4.5 is dedicated to final remarks.

4.1 The ASM, ASM^h and ASM^m Semantics Families

In this section we define a large enough family of abductive 2-valued semantics, the *affix stable model* family ASM, whose members are conservative extensions of the SM semantics. The ASM is a subfamily of the broader class of 2-valued semantics, the *fair* family. For this purpose, consider that every semantics SEM comes associated with a *reduction system*, $\mapsto_{SEM}$, that is, a set of syntactic operations that may successively reduce the original program P, by eliminating rules or by eliminating literals from the bodies, finally producing a new program P^* that keeps invariant under any further operation of the reduction system, named the *SEM remainder* of P, here denoted $remainder_{SEM}(P)$, such that $SEM(P) = SEM(P^*)$. It is implicit that this set of transformations is both *terminating* and *confluent*, meaning that for any finite ground normal logic program P the number of operations needed to reach P^* is finite, and the order in which the operations are performed is irrelevant. Let us suppose that this set of program transformations is not stronger than the system $\mapsto_{WFS}$ associated with the WFS (see section 3.2.3), meaning that the transformation of P into $remainder_{SEM}(P)$ does neither eliminate more rules, nor more literals, than the transformation of P into $remainder_{WFS}(P)$. This is a reasonable supposition, since if a semantics SEM allows for a stronger than the $\mapsto_{WFS}$ reduction system, then the program $remainder_{SEM}(P)$ may miss some of the stable models of P, which would out rule SEM as a conservative extension of the SM semantics, since every stable model contains the well-founded model, in the knowledge ordering sense [1].

Together with a semantics SEM and the corresponding reduction system $\mapsto_{SEM}$, comes the *division* (by a set of atoms) concept.

Definition 4.1. Division. Given a normal logic program P, a semantics SEM with the corresponding reduction system $\mapsto_{SEM}$, and a set of atoms A of the language of P, the *division* of P by A is the operation whose result is the program $remainder_{SEM}(P \cup A)$, denoted by P/A.

Every terminating and confluent reduction system has a corresponding division operation. We define below a "weak enough" reduction system, here designated *gracious*, that will be taken as the weakest possible reduction system for the 2-valued semantics considered in this work. It is designed with the aim of being weaker than the reduction systems associated with the SM and the MH, meaning that all the reductions it performs are a

[1]See definition 3.8 for knowledge ordering.

subset of the reductions performed by the systems $\mapsto_{WFS}$ and $\mapsto_{LWFS}$. For that purpose, we define the layered variants of the operations of success, positive reduction and failure, respectively *layered success*, *layered positive reduction* and *layered failure*, that are weaker than the former operations, meaning that where the latter ones are applicable the former ones also are.

Definition 4.2. Layered Success.[2] Let P_1 and P_2 be normal logic programs. We say that program P_2 results from P_1 by a *layered success* operation, $P_1 \mapsto_{LS} P_2$, iff P_2 is obtained by applying on P_1 a success operation, where the rule r of P_1 used to perform the operation is not in loop through the atom say $b \in Body(r)$ used to fulfill the operation.

Definition 4.3. Gracious reduction system. We dub *gracious reduction system*, denoted by $\mapsto_G$, the set of reduction operations consisting of the operations *layered success*, *layered negative reduction*, *positive reduction* and *failure*.

The proposition below is immediate after lemma 5.17.

Proposition 4.1. The gracious reduction system is terminating and confluent when applied on any finite ground normal logic program.

We now define a large family of 2-valued semantics, the *fair* family, containing among others the most important 2-valued semantics referred in this paper, and for sure a large part of interesting 2-valued semantics, so very little demanding are the conditions therein.

Definition 4.4. Fair semantics. We say that a 2-valued semantics SEM is a *fair* semantics iff

1. SEM enjoys property C_1;
2. For any normal logic program P, $SEM(P)$ is invariant under the reduction system $\mapsto_G$;
3. SEM enjoys the property of *division satisfaction*: given a normal logic program P and a segment T of P, if $M_1 \in SEM(P^{\leq T})$ and $M_2 \in SEM(P^{>T}/M_1^+)$, then $M_2 \in SEM(P)$.

Proposition 4.2. The SM and the MH are fair semantics.

Proof. Condition (1) is valid for both semantics; condition (2) is valid for both semantics, since the reduction system $\mapsto_G$ is weaker than the reduction systems $\mapsto_{WFM}$ and $\mapsto_{MH}$; condition (3) is valid for both semantics, since if $M_2 \in SM(P^{>T}/M_1^+)$ (resp. $M_2 \in MH(P^{>T}/M_1^+)$), then M_2 is a SM (resp. MH model) of P, by definition of SM and MH. □

The concept of reduction system of a semantics motivates the concept of *T-segment* support of a model, defined below.

Definition 4.5. T-segment support. Let SEM be a 2-valued fair semantics, P a normal logic program, $P^{\leq T}$ a segment of P, and $M \in SEM(P)$. We say that $M_* \in SEM(P^{\leq T})$ *T-segment supports* M iff $M \in SEM(P^{>T}/M_*^+)$.

[2]This operation was proposed by Alexandre Pinto.

We now put forward the definition of *affix stable model* family of abductive semantics, ASM, each member of it being a conservative extension of SM semantics. For that purpose we start by defining *affix stable interpretation.*

Definition 4.6. Affix stable interpretation. Let P be a finite ground normal logic program, SEM a fair semantics, and $X \subseteq (\triangle\ remainder_{SEM}(P))^u$. We say that I is an *affix stable interpretation* of P with respect to set X and semantics SEM (or simply a *SEM stable interpretation with affix X*) iff $I = WFM(P \cup X)$ and $WFM^u(P \cup X) = \emptyset$, that is, I is the only stable model of the program $P \cup X$. We name X an *affix* (or *hypotheses set*) of interpretation I.

Definition 4.7. Affix stable model semantics family, ASM. A fair semantics SEM belongs to the *affix stable model* semantics family, ASM, iff given a normal logic program P, $SEM(P)$ contains all the stable models of P, in case they exist, plus a subset (possibly empty) of the affix stable interpretations of P, chosen by resorting to specifically enounced criteria. An atom b of the language of P is true (false) under the semantics SEM iff b pertains to every (neither) model in $SEM(P)$; in other cases b is undefined under the semantics SEM.

Proposition 4.3. The SM semantics and the MH semantics belong to the ASM family.

Proof. The SM semantics corresponds to the case where the subset of affix stable model interpretations mentioned in definition 4.7 is empty. The MH semantics corresponds to the case where the criteria mentioned in definition 4.7 relies in the minimality of the affixes with respect to set inclusion, being these affixes subsets of the undefined set of atoms in $\triangle\ remainder_{SEM}(P)$. □

The two subfamilies of ASM next defined, ASM^h and ASM^m, are the classes whose formal properties we study in the remainder of this chapter. The notions of *potential* and *de facto* hypotheses of a program with respect to a semantics of the ASM family, set out below, are useful for clearer definitions of the classes ASM^h and ASM^m.

Definition 4.8. Given a semantics $SEM \in ASM$ and a normal logic program P, the set of *potential hypotheses* of P with respect to SEM, is the union of all the sets of hypotheses considered in the affixes of the stable interpretations of P, while the set of *de facto hypotheses* of P with respect to SEM, is the union of all the sets of hypotheses considered in the affixes of the models of P.

Example 4.1. Consider the SM semantics of program P,

$$\begin{aligned} a &\leftarrow not\ b \\ b &\leftarrow not\ a \\ c &\leftarrow a \\ c &\leftarrow not\ c \end{aligned}$$

where $P = remainder_{WFS}(P)$. The only stable model of P is $\{a, c\}$ with affix $\{a\}$. Thus a is the only *de facto* hypothesis of P, while the set of *potential* hypotheses is $\{a, b, c\}$.

Definition 4.9. $\mathbf{ASM^h}$ and $\mathbf{ASM^m}$ families. For any semantics SEM of the ASM^h or ASM^m families, and any normal logic program P:

1. The *potential hypotheses* are among the atoms that, being not facts, appear default negated in $remainder_{SEM}(P)$;

2. The taken models of P, are only those having non-empty minimal affixes with respect to set inclusion, plus the empty affix one (should it exist), in case $SEM \in ASM^h$; the taken models of P are minimal in the classical sense, in case $SEM \in ASM^m$.

We set forth some examples of ASM^h and ASM^m members. Besides SM, MH and others, the following are members of the ASM^h family[3] :

> $\mathbf{MH^{LS}}$: the reduction system is obtained by replacing the success operation in $\mapsto_{LWFS}$ by the layered success operation; the potential hypotheses of a normal logic program P are the atoms that, being not facts, appear default negated in the corresponding remainder.
>
> $\mathbf{MH^{Loop}}$: the reduction system is $\mapsto_{LWFS}$; the potential hypotheses of a normal logic program P are the atoms that, being not facts, appear default negated in literals involved in loops in the $remainder_{LWFS}(P)$.
>
> $\mathbf{MH^{Sustainable}}$: the reduction system is $\mapsto_{LWFS}$; the potential hypotheses of a normal logic program P are the atoms that, being not facts, appear default negated in the $remainder_{LWFS}(P)$, with the following additional condition: if H is a set of hypotheses of a $MH^{Sustainable}$ model M of P , then
>
> $$\forall_{h \in H}\ [(H \setminus \{h\}) \neq \emptyset \Rightarrow h \in WFM^u(P \cup (H \setminus \{h\}))],$$
>
> that is, no single hypothesis may be defined in the well-founded model if we join to P all the other remaining hypotheses.
>
> $\mathbf{MH^{Sustainable}_{min}}$: the reduction system is $\mapsto_{LWFS}$; retrieves the minimal models contained in $MH^{Sustainable}(P)$ for any normal logic program P. This semantics also belongs to the ASM^m family, since all models it retrieves are minimal.
>
> $\mathbf{MH^{Regular}}$: the reduction system is $\mapsto_{LWFS}$; retrieves the same models as MH, execept the irregular ones.

Besides SM and others, the following are members of the ASM^m family:

> **Navy**: the reduction system is $\mapsto_{WFS}$. Given a normal logic program P, $Navy(P)$ contains all the minimal models of $remainder_{WFS}(P)$.
>
> **Blue**: the reduction system is $\mapsto_{WFS}$. Given a normal logic program P, $Blue(P)$ contains all the models in $Navy(P \cup K)$ where K is obtained after terminating the following algorithm [4]:
> (a) Compute $K = kernel_{Navy}(remainder_{WFS}(P))$ – see definition 4.10, for a definition of $kernel$;
> (b) Compute $K' = kernel_{Navy}(remainder_{WFS}(P \cup K))$;
> (c) If $K \neq K'$, then let P be the new designation of program $P \cup K'$; go to step (a). Repeat steps (a) – (c) until $K \neq K'$ comes false in (c).

[3]The first three semantics, besides MH, were suggested by Alexandre Pinto.
[4]This algorithm is presented in [Dix95a].

Cyan: the reduction system is $\mapsto_{WFS}$. Given a normal logic program P, compute $Cyan(P)$ through the steps of *Blue* computation, but taking only the *regular* models (see definition 4.20) to compute the semantic kernel at steps (a) and (b).

Green: the reduction system is $\mapsto_{WFS}$. Given a normal logic program P, $Green(P)$ contains all the minimal models of $remainder_{WFS}(P)$ that have the smallest (with respect to set inclusion) subsets of classically unsupported atoms.

4.2 Advantageous Formal Properties

The set of formal properties we are dealing with in this work, for the purpose of characterizing some of the semantics herein defined and/or referred, are existence, relevance, cut and cautious monotony, defined below. Cut and cautious monotony, when both valid, qualifies a semantics as *cumulative*. Cumulativity belongs to the class of *strong* semantics properties, whilst relevance belongs to the class of *weak* properties [Dix95b]. For the purpose of expediting the presentation of this chapter definitions and results, we present the concept of *semantic kernel* of a normal logic program.

Definition 4.10. Semantic kernel of a normal logic program. Let P be a normal logic program and SEM a fair semantics. We define the *semantic kernel* of P with respect to SEM (or simply *kernel* of P with respect to SEM), denoted by $kernel_{SEM}(P)$, as the following subset of the Herbrand basis of P

$$kernel_{SEM}(P) = \bigcap_{M \in SEM(P)} M^+,$$

where $SEM(P) \neq \emptyset$.

Definition 4.11. Existence. (adapted from [PP11]) We say a semantics SEM is existential iff every normal logic program has at least one SEM model.

Definition 4.12. Cautious monotony. (adapted from [Dix95a]) We say a semantics SEM enjoys the property of *cautious monotony* iff for every normal logic program P and for every set $S \subseteq ker_{SEM}(P)$ we have $ker_{SEM}(P) \subseteq ker_{SEM}(P \cup S)$.

Definition 4.13. Cut. (adapted from [Dix95a]) We say a semantics SEM enjoys the property of *cut* iff for every normal logic program P and for every set $S \subseteq ker_{SEM}(P)$ we have $ker_{SEM}(P \cup S) \subseteq ker_{SEM}(P)$.

Definition 4.14. Cumulativity. (adapted from [Dix95a]) We say a semantics SEM enjoys the property of *cumulativity* iff it enjoys both cut and cautious monotony.

Definition 4.15. Relevance. (adapted from [Dix95b]) We say a semantics SEM enjoys the property of *relevance* iff for every normal logic program P we have

$$\forall_{a \in \mathcal{H}_P}(a \in ker_{SEM}(P) \Leftrightarrow a \in ker_{SEM}(Rel_P(a))),$$

where $Rel_P(a)$ is the subprogram of P relevant to atom a.

Definition 4.16. Global to local relevance. We say a semantics SEM enjoys the property of *global to local relevance* iff for every normal logic program P we have

$$\forall_{a \in \mathcal{H}_P}(a \in ker_{SEM}(P) \Rightarrow a \in ker_{SEM}(Rel_P(a))).$$

Definition 4.17. Local to global relevance. We say a semantics SEM enjoys the property of *local to global relevance* iff for every program P we have

$$\forall_{a \in \mathcal{H}_P}(a \in ker_{SEM}(Rel_P(a)) \Rightarrow a \in ker_{SEM}(P)).$$

Thus a semantics enjoys the property of relevance iff it enjoys both "global to local" and "local to global" relevance.

Notice that when we say, for instance, "the semantics SEM enjoys the property of cut", a certain subset of logic programs is implicitly being taken as the domain of the semantics, namely the subset of normal logic programs where SEM is defined. We say "stable model semantics is not cumulative" because there are examples of normal logic programs that have stable models whilst cumulativity fails for them. Anyway, it is of course correct to say "the stable model semantics is cumulative in the domain of definite logic programs [vEK76]".

4.3 Characterization of Cumulativity for ASM^h and ASM^m Families

In this section we lay down a characterization of semantics of ASM^h and ASM^m families, with respect to cumulativity, where it is shown that SEM is cumulative iff for any finite ground normal logic program P and any set $S \subseteq ker_{SEM}(P)$ we have $SEM(P) = SEM(P \cup S)$.

Cautious monotony is characterized by proposition 4.4 and corollary 4.5 below.

Proposition 4.4. Let SEM be a semantics of ASM^h or ASM^m families. SEM is not cautious monotonic iff there is a ground normal logic program P, a 2-valued interpretation M of P and a subset S of $ker_{SEM}(P)$, such that $M \in SEM(P \cup S)$ and $M \notin SEM(P)$.

Proof. See appendix B.1.

Corollary 4.5. (of proposition 4.4) A semantics SEM of the $ASM^h \cup ASM^m$ class is cautious monotonic iff, for every ground normal logic program P and for every subset $S \subseteq ker_{SEM}(P)$, it is the case that $SEM(P \cup S) \subseteq SEM(P)$.

Definition 4.12 of cautious monotony presupposes $SEM(P) \neq \emptyset$. Meanwhile, proposition 4.10, in section 4.4, states the failure of cautious monotony if existence fails for any semantics of ASM^h or ASM^m families. This means cautious monotony fails if $SEM(P) = \emptyset$ or $SEM(P \cup S) = \emptyset$.

Cut is characterized by proposition 4.6 and corollary 4.7 below.

Proposition 4.6. Let SEM be a semantics of ASM^h or ASM^m families. SEM is not cut iff there is a normal logic program P, a 2-valued interpretation M of P and a subset S of $ker_{SEM}(P)$, such that $M \in SEM(P)$ and $M \notin SEM(P \cup S)$.

Proof. See appendix B.1.

Corollary 4.7. (of proposition 4.6) A semantics SEM of the $ASM^h \cup ASM^m$ class is cut iff, for every ground normal logic program P and for every subset $S \subseteq ker_{SEM}(P)$, it is the case that $SEM(P) \subseteq SEM(P \cup S)$.

Definition 4.13 of cut, presupposes $SEM(P \cup S) \neq \emptyset$. Meanwhile, if $SEM(P) \neq \emptyset$ and $SEM(P \cup S) = \emptyset$, then cut immediately fails for any semantics of ASM^h or ASM^m families, as stated in the next proposition.

Proposition 4.8. Let SEM be a semantics of the ASM^h or ASM^m families, P a ground normal logic program such that $SEM(P) \neq \emptyset$, and $S \subseteq ker_{SEM}(P)$. If $SEM(P \cup S) = \emptyset$, then SEM is not cut.

Proof. See appendix B.1.

The failure of existence for a semantics of $ASM^h \cup ASM^m$, is not enough to establish conclusions about cut – e.g., MH is existential but is not cut, $Blue$ is existential and is cut, SM is not existential and is cut, $MH^{Sustainable}$ is not existential and is not cut.

The following theorem stems directly from the above characterizations.

Theorem 4.9. Let SEM be a semantics of ASM^h or ASM^m families. Then SEM is cumulative iff $SEM(P) = SEM(P \cup S)$, for any ground normal logic program P and any subset $S \subseteq ker_{SEM}(P)$.

Example 4.2. It is well-known that the SM semantics is cut [Dix95b]. Consider program P

$$\begin{aligned} a &\leftarrow not\ b \\ b &\leftarrow not\ a \\ c &\leftarrow a \\ c &\leftarrow not\ c \end{aligned}$$

whose only SM model is $\{a, c\}$ with affix $\{a\}$. Then $ker_{SM}(P) = \{a, c\}$. Meanwhile the SM models of $P \cup \{c\}$ are $\{a, c\}$ and $\{b, c\}$. Thus it is the case that $SM(P) \subseteq SM(P \cup \{a\})$, as would be expected, after corollary 4.7, of a ASM^m semantics that is cut.

Corollaries 4.5, 4.7 and theorem 4.9 allow us to conclude that a semantics of ASM^h or ASM^m families is not cumulative, by resorting to counter examples, even in cases where the programs used to set a counter example do not show explicitly a failure of this property. This type of study is not within the reach of common procedures to spot the failure of cumulativity (e.g. [Dix95a, Dix95b]), which generally proceed as follows: compute all the SEM models of a program P, add to P subsets $S \subseteq ker_{SEM}(P)$, and compute all the models of the resulting programs $P \cup S$, drawing a conclusion about cumulativity only in the cases where $ker_{SEM}(P) \neq ker_{SEM}(P \cup S)$. To make this point clear, consider the following examples.

Example 4.3. Let P be the 1-layer program

$$\begin{array}{lll} a \leftarrow not\ b, not\ s & d \leftarrow b & d \leftarrow a \\ b \leftarrow not\ a, not\ c & d \leftarrow not\ d & c \leftarrow k \\ c \leftarrow not\ b, not\ k & k \leftarrow a, d & s \leftarrow not\ a, d \end{array}$$

whose SM models are $\{a, c, d, k\}$ and $\{b, d, s\}$, and thus $ker_{SM}(P) = \{d\}$. Now $P \cup \{d\}$ has the stable models $\{a, c, d, k\}$, $\{b, d, s\}$ and $\{c, d, s\}$, which gives us $ker_{SM}(P) = ker_{SM}(P \cup \{d\}) = \{d\}$. Hence no negative conclusion can be taken about cumulativity by means of usual procedures. But using the result of theorem 4.9 it is immediate to conclude that the SM does not enjoy the property of cumulativity, because $SM(P) \neq SM(P \cup S)$. More specifically, corollary 4.5 tells us, via this example, that the SM is not cautious monotonic.

Example 4.4. The following 1-layer program P shows that none of the semantics MH, MH^{LS}, MH^{Loop}, $MH^{Sustainable}$ and $MH^{Regular}$ is cautious monotonic or cut.

$$\begin{array}{ll} u \leftarrow b & a \leftarrow not\ b \\ u \leftarrow c & b \leftarrow not\ c \\ t \leftarrow a & c \leftarrow h, u \\ t \leftarrow h & h \leftarrow not\ h, not\ t \end{array}$$

Let SEM represent any of these semantics. The minimal hypotheses models are the same with respect to any of the four semantics: $\{a, c, u, t\}$ with affix $\{c\}$; $\{b, c, h, u, t\}$ with affix $\{b, h\}$; $\{b, u, t\}$ with affix $\{t\}$. Thus $ker_{SEM}(P) = \{t, u\}$. Now it is the case that $remainder_{LWFS}(P \cup \{u\})$ is the same for any of these semantics,

$$\begin{array}{lll} u \leftarrow b & a \leftarrow not\ b & t \leftarrow h \\ u \leftarrow c & b \leftarrow not\ c & h \leftarrow not\ h, not\ t \\ t \leftarrow a & c \leftarrow h & u \leftarrow \end{array}$$

(as a matter of fact, the remainder for the MH^{LS} has the rule $c \leftarrow h, u$ instead of $c \leftarrow h$, but this does not change the sequel of this reasoning). The minimal hypotheses models of $remainder_{SEM}(P \cup \{u\})$ are the same with respect to any of the four semantics: $\{c, u, a, t\}$ with affix $\{c\}$; $\{h, u, c, t, a\}$ with affix $\{h\}$; $\{t, b, u\}$ with affix $\{t\}$. Thus $ker_{SEM}(P \cup \{u\}) = \{t, u\} = ker_{SEM}(P)$, and no conclusions about cumulativity can be drawn by means of the usual procedures. Meanwhile $M = \{h, u, c, t, a\}$, with affix $\{h\}$, is a minimal affix model of $P \cup \{u\}$, but is not a minimal affix model of P, which by corollary 4.5 renders any of these semantics not cautious monotonic. Also $N = \{b, c, h, u, t\}$, with affix $\{b, h\}$, is a minimal affix model of P, but not a minimal affix model of $P \cup \{u\}$, which by corollary 4.7 renders any of these semantics not cut.

It should be stressed that there are 2-valued cumulative semantics to which $SEM(P) \neq SEM(P \cup S)$ for some normal logic program P and $S \subseteq ker_{SEM}(P)$ – theorem 4.9 states this is not the case if SEM belongs to ASM^h or ASM^m families. Consider, for instance, the semantics *Picky* defined as follows: for any normal logic program P, $Picky(P) = SM(P)$ iff $ker_{SM}(P) = ker_{SM}(P \cup S)$, for every $S \subseteq ker_{SM}(P)$; otherwise $Picky(P) = \emptyset$. This semantics is cumulative, by definition, but it is not always the case that $SEM(P) =$

$SEM(P \cup S)$: for program P of example 4.3, we have $Picky(P) = \{\{a, d, c, k\}, \{b, d, s\}\}$ and $Picky(P \cup \{d\}) = \{\{a, d, c, k\}, \{b, d, s\}, \{c, d, s\}\}$. *Picky* is not a *ASM* semantics, because it does not extend conservatively the *SM* semantics. Moreover, it can be shown that *Picky* is not even a fair semantics.

4.4 Defectivity, Excessiveness and Irregularity

In this section we define three structural properties of fair semantics, *defectivity*, *excessiveness* and *irregularity*, and show that for semantics of the ASM^h or ASM^m families, defectivity is equivalent to the failure of existence and to the failure of global to local relevance, and also implies the failure of cautious monotony, whilst excessiveness implies the failure of cut, and irregularity is equivalent to the failure of local to global relevance. One of the virtues of these structural properties, is that in some cases we do not even need to compute all the models of a program to spot their validity, hence providing a shortcut to detect the failure of existence, relevance or cumulativity, as we shall see in the sequel.

4.4.1 Defectivity

The rationale of the concept of defective semantics is the following: if a normal logic program P has a segment, say $P^{\leq T}$, and a SEM model M of $P^{\leq T}$ that T-segment supports no SEM model of P, that is $SEM(P^{>T}/M^{+}) = \emptyset$, then we say the semantics is defective, in the sense that it "does not use" all the models of segment T in order to get whole models of P.

Definition 4.18. Defective semantics. A semantics SEM of the fair family is called *defective* iff there is a normal logic program P, $SEM(P) \neq \emptyset$, a segment $P^{\leq T}$ of P, and a SEM model M of the segment $P^{\leq T}$, such that M does not T-segment support any SEM model of P, that is $SEM(P^{>T}/M^{+}) = \emptyset$. We also say that SEM is *defective* with respect to segment T of program P, and that M is a *defective* model of P with respect to segment T and semantics SEM.

Example 4.5. Program P in example 4.2 may be used to show that the SM semantics is defective. The only SM model of P is $N = \{a, c\}$ with affix $\{a\}$, and $P^{\leq 1} = \{a \leftarrow \textit{not } b,\ b \leftarrow \textit{not } a\}$ has the stable model $M = \{\textit{not } a, b\}$ that does not 1-segment support N since $SM(P^{>1}/\{b\}) = \emptyset$.

It is clear from the definition that defectivity spots the non existence property of a semantics. The next result shows a tight relation between existence and defectivity, for semantics of the families ASM^h and ASM^m.

Proposition 4.10. *Defectivity* $\Leftrightarrow$ $\neg$ *Existence.* A semantics SEM of ASM^h or ASM^m families is defective iff it is non existential.

Proof. See appendix B.2.

This result allows to detect the failure of the existence property for semantics of ASM^h or ASM^m families, by resorting to counter examples, even in some cases where the programs used as counter-examples have a semantics. E.g., the program P in example 4.2 can be used to detect the failure of existence for SM semantics, in spite of the existence of stable models for program P, since it reveals the defectivity of SM. Notice that there are 2-valued semantics for which this property fails, e.g., M_P^{Supp} [ABW88] which is not defective in spite of being not existential – it is the case that M_P^{Supp} is not a ASM semantics, since it does not extend conservatively the SM semantics.

The next result shows that for semantics of the ASM^h or ASM^m families, lack of existence implies the failure of cautious monotony.

Proposition 4.11. ***$\neg$ Existence $\Rightarrow$ $\neg$ Cautious monotony.*** If a semantics SEM of the ASM^h or ASM^m families is not existential, then it is not cautious monotonic.

Proof. See appendix B.2.

Notice that there are 2-valued semantics for which this property fails, e.g., M_P^{Supp} [ABW88] which is not existential but is cautious monotonic – as referred above $M_P^{Supp} \notin ASM$.

The converse of this result, $\neg$ Cautious Monotony $\Rightarrow$ $\neg$ Existence, is not valid for the ASM^h or ASM^m families. To see this is the case, notice that on the side of ASM^h family the semantics MH is existential, although in example 4.4 it is show that this semantics is not cautious monotonic; on the side of ASM^m family, $Green$ is not cautious monotonic, since it mimics the SM behavior with respect to program in example 4.2, but is existential.

In [Dix95b], section 5.6, the author says, about a program related with the one in example 4.2, that the SM is not cumulative and that this fact does not depend on the nonexistence of stable models. Although this claim is true when considering the program in example 4.2, proposition 4.11 above shows that the SM semantics could not be cautious monotonic since it is non-existential. Thus a relation between these properties in what concerns the SM, seems to be attainable only by considering the universe of the normal logic programs, in the vein of the approach taken in this paper. It appears not to come out on a basis of individual programs analysis.

The following corollary is immediate after propositions 4.10 and 4.11.

Corollary 4.12. Defectivity $\Rightarrow$ $\neg$ Cautious monotony. If a semantics SEM of ASM^h or ASM^m families is defective, then it is not cautious monotonic.

The next two results show that defectivity is equivalent to the failure of global to local relevance, for any semantics of ASM^h or ASM^m families. Hence, due to proposition 4.10, lack of existence immediately implies lack of relevance for semantics of these types.

Proposition 4.13. ***Defectivity $\Rightarrow$ $\neg$ Global to local relevance.*** If a semantics SEM of ASM^h or ASM^m families is defective, then it fails the property of global to local relevance.

Proof. See appendix B.2.

Proposition 4.14. ***$\neg$ Global to local relevance $\Rightarrow$ Defectivity.*** If a semantics SEM of ASM^h or ASM^m families is not global to local relevant, then it is defective.

Proof. See appendix B.2.

Proposition 4.13, together with example 4.5, show that SM semantics is not global to local relevant and hence not relevant. The theorem below stems directly from the results presented above in this section.

Theorem 4.15. The following relations are valid for any semantics of the ASM^h or ASM^m families:

$$\neg\text{Existence} \Leftrightarrow \text{Defectivity} \Leftrightarrow \neg\text{Global to Local Relevance} \Rightarrow \neg\text{Cautious Monotony}$$

Proof. See appendix B.2.

4.4.2 Excessiveness and Irregularity

The rationale of the concept of *excessive semantics* is the following: if a normal logic program P has a model N that is not T-segment supported by any SEM model of $P^{\leq T}$, meaning that for every model $M_* \in SEM(P^{\leq T})$ it is the case that $N \notin SEM(P^{>T}/M_*^+)$, then we say that model N (and thus the semantics) is excessive, in the sense that it "goes beyond" the semantics of the segment $P^{\leq T}$ by not being a "consequence" of it. The concept of *irregularity* resembles excessiveness (see definition in the sequel), but they exhibit a certain independence, meaning that they can both occur, both fail, or only one of them fail in semantics of ASM^h and ASM^m families.

Definition 4.19. Excessive semantics. A semantics SEM of the fair family is called *excessive* iff there is a logic program P, a segment T of P, a model $M \in SEM(P^{\leq T})$ and a model $N \in SEM(P)$ such that:

1. $M^+ = N^+_{\leq T}$, where $N^+_{\leq T} = N^+ \cap Heads(P^{\leq T})$;
2. For every model $M_* \in SEM(P^{\leq T})$ it is the case that $N \notin SEM(P^{>T} \cup M_*^+)$;
3. There is at last a SEM model N^* of P, such that $N^* \in SEM(P^{>T} \cup M^+)$.

Example 4.6. The following program P shows that semantics MH, MH^{LS}, MH^{Loop}, $Navy$ and $Green$ are excessive,

$$\begin{gathered} a \leftarrow not\ b \\ b \leftarrow not\ a \\ -----1 \\ u \leftarrow a \\ u \leftarrow b \\ -----2 \\ p \leftarrow not\ p, not\ u \\ -----3 \\ q \leftarrow not\ q, not\ p \\ -----4. \end{gathered}$$

In fact $N = \{a, u, p, not\ b, not\ q\}$, with affix $\{a, p\}$, is a model of P under any of these semantics, and no SEM model of $P^{\leq 2}$, $\{a, not\ b, u\}$ and $\{not\ a, b, u\}$, 2-segment supports N, i.e., for every model $M_* \in SEM(P^{\leq 2})$ it is the case that $N \notin SEM(P^{>2}/M_*^+)$, because the division "/" eliminates the rule in layer 3 via layered negative reduction operation, for any $SEM \in \{MH, MH^{LS}, MH^{Loops}, Navy, Green\}$.

Definition 4.20. Irregular semantics. A semantics SEM of the fair family is called *irregular* iff there is a normal logic program P, a segment T of P and a SEM model N of P, such that for no model M of $P^{\leq T}$ do we have $N^+_{\leq T} = M^+$, where $N^+_{\leq T} = N^+ \cap Heads(P^{\leq T})$. We also say that SEM is *irregular* with respect to segment T of program P, and that N is an *irregular* model of P with respect to segment T and semantics SEM. A model that is not irregular is called *regular*, and a semantics that produces only regular models is called *regular*.

The following examples expose the independence of the concepts of excessiveness and irregularity for semantics of the class $ASM^h \cup ASM^m$, meaning the existence of semantics in this class for any of the four possible cases of validity or failure of excessiveness and irregularity.

Example 4.7. Program P below shows that the semantics MH, MH^{LS} and MH^{Loop}, $Green$, $Navy$ and $Blue$ are all irregular.

$$
\begin{aligned}
& a \leftarrow not\ b \\
& b \leftarrow not\ a \\
& -----1 \\
& p \leftarrow not\ p, not\ a \\
& q \leftarrow not\ q, not\ b
\end{aligned}
$$

In fact all these semantics admit the model $N = \{a, b, not\ p, not\ q\}$. The models of segment $P^{\leq 1}$ are $\{a, not\ b\}$ and $\{b, not\ a\}$, none of whose positive sets of atoms equals $N^+_{\leq T} = \{a, b\}$. As $Blue$ is not excessive, this example shows *irregularity* $\not\Rightarrow$ *excessiveness*.

To see that there are excessive though not irregular semantics, consider the semantics $MH^{Regular}$ defined as follows: for any normal logic program P, $MH^{Regular}(P)$ has the same models as $MH(P)$ except the irregular ones.

Example 4.8. The following program P shows that $MH^{Regular}$ may generate excessive models but not irregular ones.

$$
\begin{aligned}
& e \leftarrow not\ e \\
& -----1 \\
& b \leftarrow not\ b, not\ a, not\ e \\
& a \leftarrow not\ a, not\ b
\end{aligned}
$$

In fact, the $MH^{Regular}$ models of P are $\{e, a\}$, with affix $\{e, a\}$ and $\{e, b\}$, with affix $\{e, b\}$. Now $\{e, b\}$ is excessive, since $\{e, b\} \notin MH(P^{>1}/\{e\})$, but it is not irregular, by design of $MH^{Regular}$ – notice that both models are regular. Thus *excessiveness* $\not\Rightarrow$ *irregularity*.

Examples 4.6 and 4.7 show that MH, for instance, is excessive and irregular. Meanwhile, $Cyan$ is not excessive and is not irregular. The following result tells us that a stable model is neither excessive nor irregular, and hence SM is a regular semantics.

Proposition 4.16. Let P be a normal logic program and $M \in SM(P)$. Then M is neither excessive nor irregular.

Proof. See appendix B.2.

The ensuing result shows that for any semantics of ASM^h or ASM^m families, excessiveness implies the failure of cut.

Proposition 4.17. ***Excessiveness* $\Rightarrow \neg$ *Cut.*** If a semantics SEM of ASM^h or ASM^m families is excessive, then it is not cut.

Proof. See appendix B.2.

This result, together with example 4.6, shows that semantics MH, MH^{LS}, MH^{Loop}, $Navy$ and $Green$ are not cut.

The converse of this result, $\neg$ Cut $\Rightarrow$ Excessiveness, is not valid for semantics of the families ASM^h or ASM^m. For example, $MH^{Sustainable} \in ASM^h$ is not cut and not excessive. The same for the semantics $MH^{Sustainable}_{min} \in ASM^m$, where $MH^{Sustainable}_{min}(P)$ retrieves the minimal models contained in $MH^{Sustainable}(P)$ for any normal logic program P.

The result that follows states that irregularity is equivalent to the failure of local to global relevance, for semantics of ASM^h or ASM^m families.

Proposition 4.18. ***Irregularity* $\Leftrightarrow \neg$ *Local to global relevance.*** A semantics SEM of ASM^h or ASM^m families is irregular iff it is not local to global relevant.

Proof. See appendix B.2.

This result, together with the program in example 4.7, shows that semantics MH, MH^{LS} and MH^{Loop}, $MH^{Sustainable}$, $Green$, $Navy$ and $Blue$ fail local to global relevance, and are thus not relevant. The following theorem stems directly from the statements of propositions 4.17 and 4.18.

Theorem 4.19. The following relations stand for any semantics of ASM^h or ASM^m families.

$$\begin{aligned} \text{Excessiveness} &\Rightarrow \neg\text{Cut} \\ \text{Irregularity} &\Leftrightarrow \neg\text{Local to Global Relevance} \end{aligned}$$

It should be noticed that for the class $ASM^h \cup ASM^m$:

- Defectivity and cut are unrelated properties: $MH^{sustainable}$ is defective and is not cut; MH is not defective and is not cut; $Blue$ is not defective and is cut; SM is defective and is cut.

- Excessiveness and cautious monotony are unrelated properties: SM is not cautious monotonic and is not excessive; MH is not cautious monotonic and is excessive; $Navy$ is cautious monotonic and is excessive; $Blue$ is cautious monotonic and is not excessive.

- Irregularity and cautious monotony are unrelated properties: MH is irregular and is not cautious monotonic; SM is not irregular and is not cautious monotonic; $Navy$ is irregular and is cautious monotonic; $Cyan$ is not irregular and is cautious monotonic.

4.5 Final Remarks

We have presented in this chapter a study on the formal properties of 2-valued conservative extensions of the SM semantics, under a structural point of view. For that purpose we defined a large class of 2-valued semantics, the fair family, and took a particular subclass of this family, the affix stable model semantics, ASM, where all the semantics pertaining to the ASM family are 2-valued conservative extensions of the SM semantics. Having defined the ASM family, we focused on two subsets of ASM, the classes ASM^h and ASM^m. Our intention is to characterize semantics of these two classes on the properties of existence, relevance and cumulativity through the analysis of the decomposition of models with respect to the layers of the programs. This point of view reveals itself advantageous on various aspects, when compared to common processes for characterizing semantics on these properties, as we have shown in this chapter: (1) It furnishes a larger universe of normal logic programs that may be used as counter-examples to spot the failure of one or more of these properties – e.g. corollaries 4.5, 4.7; (2) It reveals relations among the properties that allow to draw conclusions on some of them on basis of held knowledge about others – e.g. theorems 4.15 and 4.19. This last point builds on top of the structural properties of defectivity, excessiveness and irregularity, that in certain cases require only the computation of a couple of models of a program to draw conclusions about the above referred formal properties over the universe of normal logic programs, providing an alternative to the sometimes harsh path of direct proofs.

If we consider the five formal properties of existence ($\exists$), global to local relevance (gl), local to global relevance (lg), cautious monotony (cm) and cut (cut), the validity or failure of each of these properties allow the existence of $2^5 = 32$ types of semantics. Meanwhile, the study we present in this work shows that only 12 such types of semantics may exist in the classes ASM^h and ASM^m. They are represented in table 4.1, where 0 flags the failure of a property, and 1 means the property is verified. The 20 missing semantics correspond to cases where ($\exists = 0$ and $gl = 1$), or ($\exists = 1$ and $gl = 0$), or ($\exists = 0$ and $cm = 1$), each of these cases going against the statement of theorem 4.15. The correspondence of ASM^h and ASM^m semantics presented in this text and the entries in table 4.1 is as follows (see appendix B.3): 1. $MH^{sustainable}, MH^{Sustainable}_{min}$ 2. $--$ 3. $--$ 4. SM 5. $MH, MH^{LS}, MH^{Loop}, Green$ 6. $--$ 7. $Navy$ 8. $Blue$ 9. $MH^{Regular}$ 10. $--$ 11. $--$ 12. $Cyan$. Whether semantics of $ASM^h \cup ASM^m$ exist for the types marked with "$--$", may be taken as an open issue.

It is worth to notice that in the definitions of $MH^{Regular}$ and $Cyan$, local to global relevance

Table 4.1: The 12 possible types of ASM^h and ASM^m semantics

	$\exists$	gl	lg	cm	cut
1	0	0	0	0	0
2	0	0	0	0	1
3	0	0	1	0	0
4	0	0	1	0	1
5	1	1	0	0	0
6	1	1	0	0	1
7	1	1	0	1	0
8	1	1	0	1	1
9	1	1	1	0	0
10	1	1	1	0	1
11	1	1	1	1	0
12	1	1	1	1	1

is gotten by excluding all irregular models. This is only possible because the structural property of irregularity is equivalent to the failure of the formal property of local to global relevance, when we consider the ASM^h and ASM^m families. The point here is that structural properties, as defined in this work, characterize models: we have defective models, excessive models and irregular models. There is thus a path of investigation put forward by this work, whose goal is to allow the design of semantics which enjoy some desired properties, by excluding the models responsible for the failure of those very same properties. Were this possible with respect to cautious monotony and cut properties in the ASM^h and ASM^m families, for example, and then cumulative semantics could be obtained by other means than that of an iterative procedure (see suggestion 12 for future work in chapter 7).

The above research reveals a "syntactic" nature of the studied formal properties, and clearly opens the possibility to tackle from a syntactic perspective other *strong* and *weak* properties of semantics of logic programs. It also proffers a path of investigation to extend/redefine the properties of defectivity, excessiveness and irregularity to n-valued logic program semantics, $n \neq 2$.

Chapter 5

A Paraconsistent Extension of the Minimal Hypotheses Semantics, MH_P

In this chapter we present a paraconsistent abdutive semantics for extended normal logic programs, the paraconsistent minimal hypotheses semantics MH_P. Part of the contents o this chapter is based upon our previous publication [AP13].

The MH_P is a semantics of *total paraconsistent models* which combines the merits of two already existing semantics: it inherits the existence property of the abductive *minimal hypotheses* semantics MH [PP11], which is a semantics of total 2-valued models, and the property of detection of support on contradiction of the *paraconsistent well-founded semantics with explicit negation* $WFSX_P$ [ADP95], which is a semantics of *partial paraconsistent models.* The MH_P enjoys also the property of *simple relevance* (definition 3.18 can be transcrited for extended normal logic programs), which permits top-down query answering for brave reasoning purposes. Besides, the MH_P lends itself to various types of skeptical and brave reasoning, which include the possibility of drawing conclusions from default inconsistent models in a nontrivial way. The MH_P coincides with the MH on normal logic programs, and with the $WFSX_P$ on stratified extended programs.

The rest of this chapter goes as follows. In section 5.1 we add some terminology for extended normal logic programs. Section 5.2 contains a characterization of the $WFSX_P$ semantics, as embedded into the well-founded semantics by means of the $t-o$ *transformation* there defined. In section 5.3 we describe the reduction system $\mapsto_{bLWFS}$, that takes the $t-o$ transformed P^{t-o} of an extended normal logic program P, and retrieves the *balanced layered remainder* bP^{t-o} of P. In section 5.4 we define the MH_P semantics, present the logic Six, that provides a truth-functional model theory to the MHp semantics, characterize the MH_P on the properties of existence, relevance, cumulativity and simple relevance, and finally present a result on the computational complexity of the MH_P. In section 5.5 we propose some forms of query answering for databases defined in the extended normal logic programs language.

The main results in this chapter will be stated for finite ground extended normal logic programs.

5.1 Terminology of Extended Normal Logic Programs

Definition 5.1. Extended normal logic program, ELP. (adapted from [ADP95]) An *extended normal logic program* defined over a language $\mathcal{L}$ is a finite set of ground *extended rules* (or simply *rules*, for short), each one of the form

$$l_0 \leftarrow l_1, \cdots, l_m, not\ l_{m+1}, \cdots, not\ l_n, \tag{5.1}$$

where l_i, $0 \leq i \leq n$, is an *objective literal* (either an atom b or its explicit negation, $\neg b$, where $\neg\neg b = b$); m, n are integer non negative numbers; the operator "," stands for the classical conjunctive connective and the operator "*not*" stands for default negation (*not l* is called a *default literal*). A literal (program) is *ground* if it does not contain variables. The set of all ground objective literals involved in an extended normal logic program, plus their explicit negations, is named the *(extended) Herbrand base* of P, denoted $\mathcal{H}_P$. If $m = n = 0$ a rule is called *fact*. A program containing no explicitly negated literals is called *normal* program; a rule with no explicitly negated literals is called *normal* rule. Given a rule $r = l_0 \leftarrow l_1, \cdots, l_m, not\ l_{m+1}, \cdots, not\ l_n$, the objective literal l_0 is the *head* of the rule and $l_1, \cdots, l_m, not\ l_{m+1}, \cdots, not\ l_n$ is the *body* of the rule.

The notions of *interpretation*, *satisfaction* and *model*, in the case of extended normal logic programs, are the same as for normal logic programs (see definitions 3.1, 3.2, 3.3). We say that an interpretation I is *default consistent*, or simply *consistent*, iff $I^+ \cap I^- = \emptyset$. We say that an interpretation I is *contradictory* iff for some pair of literals $b, \neg b$ we have $b \in I^+$ and $\neg b \in I^+$. An interpretation I is *coherent* [DP98] iff for every explicit literal $b \in I^+$ we have $\neg b \in I^-$.

5.2 The WFSXp Semantics

The $WFSXp$ is a paraconsistent well-founded semantics for extended normal logic programs. The $WFSX_P$ model of an extended normal logic program may be computed by means of a dedicated fixpoint operator [ADP95], as expounded in subsection 5.2.1 for purposes of contextualization. In subsection 5.2.2, however, we present a definition of the $WFSX_P$ by means of a program transformation for extended normal logic programs, here dubbed $t - o$ *transformation*, which embeds the $WFSX_P$ into the WFS. This means the $WFSX_P$ model of an extended normal logic program P, denoted by $WFM_P(P)$, may be extracted from the well-founded model of the transformed program P^{t-o}. The $t - o$ transformation is used to define the MH_P semantics, as will be seen in the sequel.

The $WFSX_P$ envisages default negation and explicit negation necessarily related through the *coherence principle* [PA92]: if $\neg b$ holds, then *not* b should also hold (similarly, if b then *not* $\neg b$).

5.2.1 Alternating Fixpoint Definition of the $WFSX_P$

We summarize the main aspects of the operator $\Gamma\Gamma_S()$, whose least fixpoint is used to retrieve the $WFSX_P$ model of a program. This operator takes as arguments the empty set and an extended normal logic program P, and it operates similarly to the composition of two Gelfond-Lifschitz Γ operators (see definition 3.11 for "Γ" operator.

To impose the coherence principle, the authors in [ADP95] resort to the semi-normal version of an extended logic program, defined as follows.

Definition 5.2. Semi-normal version of a program. (adapted from [ADP95]) The *semi-normal* version of an extended normal logic program P is the program P_S obtained from P by adding to the (possibly empty) body of each rule $r \in P$ the default literal $not\ \neg Head(r)$.

Using the semi-normal version of an extended normal logic program, a new non-monotonic operator, Γ_S, may be defined.

Proposition 5.1. Γ_S *operator, [ADP95].* The operator Γ_S is equivalent to the Γ operator over the semi-normal version P_S of an extended normal logic program P.

Proposition 5.2. *Monotonicity of* $\Gamma\Gamma_S$*, [ADP95].* The operator $\Gamma\Gamma_S$ is monotonic under set-inclusion, for arbitrary sets of objective literals.

The $WFSX_P$ model of an extended normal logic program can be defined by means of the fixpoint operator $\Gamma\Gamma_S()$, as stated by the next theorem.

Theorem 5.3. Alternating fixpoint definition of WFSXp, [ADP95]. Let P be an extended logic program whose least fixpoint of $\Gamma\Gamma_S$ is T. Then the paraconsistent well-founded model of P is $WFMp(P) = T \cup not\ (\mathcal{H}_P - \Gamma_S T)$.

The next result states that the WFM_P and the WFM of normal logic programs are the same, if we neglect the default literals involving explicitly negated atoms in the WFM_P.

Theorem 5.4. (adapted from [ADP95]) If P is a normal logic program, then the models $WFM(P)$ and $WFM_P(P)$ are equal, if we neglect the default literals involving explicitly negated atoms in the $WFM_P(P)$.

5.2.2 The $t - o$ Transformation

In [DP97] the authors present a program transformation for extended normal logic programs, there dubbed $T - TU$ *transformation*, that embeds the $WFSX_P$ into the WFS. We present a variant definition of this transformation, here dubbed $t - o$ *transformation*, that alters the notation of the $T - TU$ transform, for simplicity purposes, whilst keeping its intended meaning.

Definition 5.3. $\mathbf{t-o}$ Transformation. (adapted from [DP97]) The *$t-o$ transformation*, maps an extended normal logic program P into a normal logic program P^{t-o}, by means of the two following steps:

1. Every explicitly negated literal in P, say $\neg b$, appears also in the transformed program, where it must be read as a new atom, say $\neg\_b$. Let P^* be the program resulting from making these transformations on P.

2. Every rule $r = (Head(r) \leftarrow Body(r))$ in P^* is substituted by the following pair of rules: (i) A rule also designated r, for simplicity, obtained from $r \in P^*$ by placing the superscript "o" in the atoms of default negations that appear in $Body(r)$; (ii) A rule r^o, obtained from $r \in P^*$ by adding to $Body(r)$ the literal $not\ \neg Head(r)$ (where $\neg\neg l = l)$[1], and by placing the superscript "o" in $Head(r)$ and in every non default negated literal of $Body(r)$.

We call *co-rules* to each pair r, r^o of rules in P^{t-o} (each rule of the pair is the *co-rule* of the other one), and *co-atoms* to each pair b, b^o of atoms in P^{t-o} (each atom of the pair is the *co-atom* of the other one). To each atom of the language of P^*, there corresponds the pair of co-atoms l, l^o of the language of P^{t-o} and vice-versa.

Example 5.1. (adapted from [DP98]) The $t-o$ transformed of the program P

$$\begin{aligned} a &\leftarrow \neg c, not\ b \\ b &\leftarrow \neg a, not\ c, not\ \neg b \\ \neg a &\leftarrow \end{aligned}$$

is the program P^{t-o}:

$$\begin{aligned} a &\leftarrow \neg c, not\ b^o & a^o &\leftarrow \neg c^o, not\ b, not\ \neg a \\ b &\leftarrow \neg a, not\ c^o, not\ \neg b^o & b^o &\leftarrow \neg a^o, not\ c, not\ \neg b \\ \neg a &\leftarrow & \neg a^o &\leftarrow not\ a \end{aligned}$$

In what follows we adopt the conventions below concerning notation and terminology.

- In the notations $r = (b \leftarrow A, not\ C^o)$, $s^o = (d^0 \leftarrow E^o, not\ F, not\ \neg d)$, A and F (resp. C^o and E^o) represent conjunctions of non superscript (resp. superscript) atoms, and $not\ C^o$ (resp. $not\ F$) represents the conjunction of the default negations of all atoms appearing in C^o (resp. F).

- Given a pair of co-rules $r, r^o \in P^{t-o}$, we also call co-rules to the rules into which r, r^o are transformed by any transformation operation on P^{t-o} defined along this text, for simplicity, as long as none of the rules is erased.

- We call *double loop* to the $t-o$ transformed of a loop.

The rules with the superscript "o" in the heads are used to derive the literals that are true or undefined in the $WFMp$ model, and the rules with no superscript in the heads are used to derive the true literals of the $WFMp$ model. This is stated in the next theorem.

[1] The literal $not\ \neg Head(r)$ is added to enforce the coherence principle.

Theorem 5.5. WFSXp is embeddable into WFS. (adapted from [DP98]) Let P be an extended normal logic program. The following equivalences hold for an arbitrary objective literal L of the language of P:

- $L \in WFMp(P)$ iff $L \in WFM(P^{t-o})$;
- $not\ L \in WFMp(P)$ iff $not\ L^o \in WFM(P^{t-o})$;
- $\neg L \in WFMp(P)$ iff $\neg L \in WFM(P^{t-o})$;
- $not\ \neg L \in WFMp(P)$ iff $not\ \neg L^o \in WFM(P^{t-o})$;

where the symbols $\neg L, \neg L^o$ on the right sides of the "iffs" must be taken as names of atoms (they are not explicitly negated literals), in accordance with point 1 of definition 5.3

The below defined operator "$\bigtriangledown$", will be handy in the sequel. It computes a paraconsistent model from a 3-valued interpretation I, containing superscript and non superscript atoms, using the lexical correspondences stated in theorem 5.5.

Definition 5.4. $\bigtriangledown$ operator. Given a 3-valued interpretation $I = \langle I^+, I^u, I^- \rangle$, where I^+, I^u, I^- may contain 'o' superscript or otherwise nonsuperscript atoms, we denote by $\bigtriangledown I$ the interpretation obtained by means of the equivalences stated in theorem 5.5, substituting $WFM(P^{t-o})$ for I and $WFM_p(P)$ for $\bigtriangledown I$.

The $t - o$ transformed of a noetherian extended normal logic program is a noetherian normal logic program, as shown bellow.

Proposition 5.6. Let P^{t-o} be the $t-o$ transformed of an extended normal logic program P. Then P is noetherian iff P^{t-o} is noetherian.

Proof. Let $P^t \subseteq P^{t-o}$ (resp. $P^o \subseteq P^{t-o}$) be the set of rules with non superscript (resp. superscript) heads in P^{t-o}. Discarding the superscripts "o" in P^{t-o} atoms, and the default negations of explicitly negated literals added to the bodies of rules in P^o, the sets of rules in P, P^t and P^o are exactly the same, as per definition 5.3. Thus if there is an infinite descending chain of dependent rules in any of these three programs, then there must also be such a chain in any one of the other two programs. $\square$

No transformation process envisaged in this work changes the noetherian character of a logic program.

5.2.3 Characterization of the $WFSX_P$ Model

We present a characterization of the $WFSX_P$ model of an extended normal logic program P, $WFM_P(P)$, via the $t - o$ transformed of P.

The next proposition and corollary characterize the valuations of the pairs of co-atoms b, b^o of the language of P^{t-o}, with respect to the $WFM(P^{t-o})$: it is the case that $b^o \leq_t b$ for any such pair (b^o, b standing here for their valuations with respect to the $WFM(P^{t-o})$), where $\leq_t$ is the truth ordering (see definition 3.7 for truth ordering).

Proposition 5.7. Let P^{t-o} be the $t-o$ transformed of a finite ground extended normal logic program P. Let $\Gamma^n_{P^{t-o}}(\emptyset)$ be the result of the n^{th} self composition of the Gelfond-Lifschitz Γ operator [GL88] on program P^{t-o}, with argument $\emptyset$. Then for every $n \in \mathbb{N}$ and for every atom b of the Herbrand base of P we have

$$b^o \in \Gamma^n_{P^{t-o}}(\emptyset) \Rightarrow b \in \Gamma^n_{P^{t-o}}(\emptyset). \tag{5.2}$$

where $\Gamma^n_{P^{t-o}}(\emptyset)$ is the least model of the Gelfond-Lifschitz transformation $\frac{P^{t-o}}{\Gamma^{n-1}_{P^{t-o}}(\emptyset)}$, being $\Gamma^0_{P^{t-o}}(\emptyset) = \emptyset$.

Proof. See appendix C.1.

The following result is immediate after proposition 5.7.

Corollary 5.8. (of proposition 5.7) Let P^{t-o} be the $t-o$ transformed of an extended normal logic program P. Let b, b^o be two co-atoms of the language of P^{t-o}. Then it is not possible to have any of the following three types of valuations with respect to the $WFM(P^{t-o})$:

(a) $b = -$ $\quad b^o = +$;

(b) $b = -$ $\quad b^o = u$;

(c) $b = u$ $\quad b^o = +$;

the only possible valuations being

(d) $(b = -)$ and $(b^o = -)$, or $(b = u)$ and $(b^o = -)$, or $(b = +)$ and $(b^o = -)$;

(e) $(b = u)$ and $(b^o = u)$, or $(b = +)$ and $(b^o = u)$;

(f) $(b = +)$ and $(b^o = +)$.

The following result relates undefined literals in the $WFM_P(P)$ with pairs of undefined co-atoms in the $WFM(P^{t-o})$.

Corollary 5.9. (of theorem 5.5) Let P^{t-o} be the $t-o$ transformed of an extended normal logic program P and $WFMp(P) = \triangledown WFM(P^{t-o})$ the paraconsistent well-founded model of P. Let also b be an atom of the language of P. Then $b \in WFM^u_P(P)$ iff $b, b^o \in WFM^u(P^{t-o})$.

Proof. Immediate after theorem 5.5. □

Definition 5.5. t − o remainder. Given an extended normal logic program P, we call *$t-o$ remainder* of P to the remainder [2] of the normal logic program P^{t-o}, and denote it by $\widehat{P^{t-o}}$.

The next result shows that given a rule $r^o \in \widehat{P^{t-o}}$, the co-rule r is not erased from $\widehat{P^{t-o}}$.

[2]See subsection 3.2.3 for remainder of a normal logic program.

Proposition 5.10. Let P^{t-o} be the $t-o$ transformed of an extended normal logic program P and $\widehat{P^{t-o}}$ its $t-o$ remainder. For every rule $r^o \in \widehat{P^{t-o}}$ its co-rule r in $\widehat{P^{t-o}}$ is either an empty body rule or a nonempty body rule, i.e., r is not deleted by the computation of $\widehat{P^{t-o}}$.

Proof. Let $r^o = (b^0 \leftarrow A^o, not\ C, not\ \neg b)$ be a rule of $\widehat{P^{t-o}}$, and $r = (b \leftarrow A, not\ C^o)$ its co-rule. Then there is (1) neither an atom $a^o \in A^o$ such that $a^o \in WFM^-(P^{t-o})$, (2) nor an atom $c \in C$ such that $c \in WFM^+(P^{t-o})$, otherwise r^o would have been deleted from $\widehat{P^{t-o}}$. By (1), (2) and corollary 5.8, there is neither an atom $a \in A$ such that $a \in WFM^-(P^{t-o})$, nor an atom $c^o \in C^o$ such that $c^o \in WFM^+(P^{t-o})$. Thus the co-rule r cannot be erased by the remainder computation, because the process does not falsify the body of r. $\square$

The following proposition states that for every atom $b \in WFM^u_P(P)$ there is a pair of nonempty body co-rules in $\widehat{P^{t-o}}$ whose heads are b, b^o.

Proposition 5.11. Let P^{t-o} be the $t-o$ transformed of an extended normal logic program P and $\widehat{P^{t-o}}$ its remainder. Given an atom $b \in WFM^u_P(P)$, there is at least a pair of co-rules $r, r^o \in \widehat{P^{t-o}}$, with nonempty bodies, such that $head(r) = b$ and $head(r^o) = b^o$.

Proof. Immediate by corollary 5.9 and proposition 5.10. $\square$

The proposition below says that for each pair of nonempty body co-rules r, r^o in $\widehat{P^{t-o}}$, there is a pair of nonempty body co-rules s,s^o that r,r^o depend on. In particular the co-rules r,r^o may depend on each other.

Proposition 5.12. For each pair of co-rules $r, r^o \in \widehat{P^{t-o}}$ that have nonempty bodies, there is a pair of co-rules $s, s^o \in \widehat{P^{t-o}}$, also having nonempty bodies, with say $Head(s) = c$ and $Head(s^o) = c^o$, such that c appears in one of the bodies of the rules r, r^o, while c^o appears in the body of the other rule of the pair, and $c \in WFM^u_P(P)$. In particular, we may have $r = s$ and $r^o = s^o$.

Proof. Let $r = (b \leftarrow D, not\ F^o)$ and $r^o = (b^o \leftarrow D^o, not\ F, not\ \neg b)$ be a pair of nonempty body co-rules in $\widehat{P^{t-o}}$. Now there is (1) an atom $d \in D$ such that $d \in WFM^u(P^{t-o})$, or (2) an atom $f^o \in F^o$ such that $f^o \in WFM^u(P^{t-o})$, otherwise one of the rules, or both of them, would either be eliminated from $\widehat{P^{t-o}}$, or else have an empty body. By corollary 5.8 and by being $Body(r)$ and $Body(r^o)$ undefined with respect to the $WFM(P^{t-o})$, we have that in case (1) $d^o \in WFM^u(P^{t-o})$ and in case (2) $f \in WFM^u(P^{t-o})$, which renders at least a pair of undefined co-atoms appearing in the bodies of the two rules. By theorem 5.5 it is $d \in WFM^u_P(P)$ or $f \in WFM^u_P(P)$. This, together with proposition 5.11, means that there is a pair of co-rules $s, s^o \in \widehat{P^{t-o}}$, with heads d, d^o or f, f^o and nonempty bodies, where one of the rules r, r^o depends on s and the other on s^o. $\square$

The next proposition states an equivalence between the existence of double loops in $\widehat{P^{t-o}}$ and the existence of undefined atoms in $WFM^u_P(P)$.

Proposition 5.13. Let P^{t-o} be the $t-o$ transformed of an extended normal logic program P, and $\widehat{P^{t-o}}$ its $t-o$ remainder. Then there is an undefined literal in $WFM_P(P)$ iff $\widehat{P^{t-o}}$ contains a double loop.

Proof.
"$\Rightarrow$"
Let $b \in WFM_P^u(P)$. By corollary 5.9 and proposition 5.11, there is a pair of co-atoms $b, b^o \in WFM^u(P^{t-o})$ and a pair of co-rules with nonempty bodies, $r, r^o \in \widehat{P^{t-o}}$, where $Head(r) = b$, $Head(r^o) = b^o$. By proposition 5.12 there is a pair of co-rules $r_1, r_1^o \in \widehat{P^{t-o}}$ with nonempty bodies, where, say $Head(r_1) = d, Head(r_1^o) = d^o$, $d \in WFM_P^u(P)$ and r depends on one of the rules r_1, r_1^o, while its co-rule r^o depends on the other one. Doing with atoms d, d^o the same reasoning that we have done with b, b^o, we find another pair of co-rules r_2, r_2^o, that r_1, r_1^o depend on, and whose heads refer to a literal that is undefined in the $WFM_P(P)$. Keeping this reasoning going on, we end up with two descending chains defined by the set of rules $\{r_1, r_1^o, r_2, r_2^o, r_3, r_3^o, \cdots\}$, which cannot be infinitely descendent, by proposition 5.6. Thus, two certain co-rules of the descending chains must be equal to another pair of co-rules located above in the chain in order to prevent its infinite descent. This means that we have a double loop in $\widehat{P^{t-o}}$.
"$\Leftarrow$"
If there is a double loop in $\widehat{P^{t-o}}$, then there is at least a pair of co-rules r, r^o whose bodies are not empty in $\widehat{P^{t-o}}$, where $Head(r), Head(r^o) \notin WFM_P^+(P)$, otherwise the rules would not take part in a loop. Thus $Head(r), Head(r^o) \in WFM^u(P^{t-o})$ which, by theorem 5.5, means that $Head(r) \in WFM_P^u(P)$. □

Corollary 5.14. (of proposition 5.13) Let P be a normal logic program and $\widehat{P^{t-o}}$ its $t-o$ remainder. Then to every undefined literal in $WFM_P(P)$, say b, there is a pair of co-rules, r, r^o in $\widehat{P^{t-o}}$, $Head(r) = b$, that belong to a double loop or depend on a double loop, the dependency meaning that r, r^o are part of descending chains that involve rules of a double loop.

Proof. Immediate after the proof of proposition 5.13. □

5.2.4 Logic Slack via the $t-o$ Transformation

A truth-functional model theory for the $WFSX_P$ is provided by the 9-valued logic *Nine* [DP98]. Table 5.1 (subsection 5.4.3) represents all the possible 4-tuple valuations of a literal $(l^o, l, \neg l^o, \neg l)$ with respect to the $WFM(P^{t-o})$. The $t-o$ transformation creates a language whose theories may acquire meaning through a 20-valued logic, that we call *Twenty*, each of these values corresponding to one of the 20 entries of table 5.1. Meanwhile, these 20 entries of the table correspond to only 9 values of logic *Nine*. For this reason we say that the $t-o$ transformation introduces a *logic slack* in *Nine* with respect to logic *Twenty*. As we show in the example below, this means the "$\bigtriangledown\bigtriangleup$ – reading" of $\widehat{P^{t-o}}$ keeps invariant if some rules of P^{t-o} are erased or some facts are asserted.

Example 5.2. WFM$_\mathbf{P}$ computation Let P be the program

$$\begin{aligned} a &\leftarrow b \\ b &\leftarrow not\ c \\ c &\leftarrow not\ a \\ \neg a &\leftarrow \\ p &\leftarrow not\ p \end{aligned}$$

whose $t-o$ transformed is P^{t-o}:

$$\begin{array}{rlrl}
a & \leftarrow b & a^o & \leftarrow b^o, not\ \neg a \\
b & \leftarrow not\ c^o & b^o & \leftarrow not\ c, not\ \neg b \\
c & \leftarrow not\ a^o & c^o & \leftarrow not\ a, not\ \neg b \\
\neg a & \leftarrow & \neg a^o & \leftarrow not\ a \\
p & \leftarrow not\ p^o & p^o & \leftarrow not\ p, not\ \neg p
\end{array}$$

The $WFM_P(P)$, computed using the alternating fixpoint procedure stated in theorem 5.3, is

$$WFM_P(P) = \langle \{\neg a, c\}^+, \{p\}^u, \{a, b, \neg b, \neg c, \neg p\}^- \rangle,$$

while the $WFM(P^{t-o})$ is

$$WFM(P^{t-o}) = \langle \{\neg a, c\}^+, \{p, p^o, a, \neg a^o, b, c^o\}^u, \{a^o, b^o, \neg b, \neg b^0, \neg c, \neg c^0, \neg p, \neg p^0\}^- \rangle.$$

To see this is the case, with respect to the $WFM(P^{t-o})$, notice that the remainder $\widehat{P^{t-o}}$ is (the eliminated literals and rules are striped out),

$$\begin{array}{rlrl}
\boldsymbol{a} & \leftarrow \mathbf{b} & \cancel{a^o} & \cancel{\leftarrow b^o, not\ \neg a} \\
\mathbf{b} & \leftarrow \mathbf{not\ c^o} & \cancel{b^o} & \cancel{\leftarrow not\ c, not\ \neg b} \\
c & \leftarrow \cancel{not\ a^o} & \mathbf{c^o} & \boldsymbol{\leftarrow not\ a}, \cancel{not\ \neg b} \\
\neg a & \leftarrow & \neg a^o & \leftarrow not\ a \\
p & \leftarrow not\ p^o & p^o & \leftarrow not\ p, \cancel{not\ \neg p}
\end{array}$$

Now consider the rules in bold in program $\widehat{P^{t-o}}$. The heads of these rules involve atoms that, neglecting the superscripts, are defined atoms in the $WFMp(P)$ model: $(a = -)$, $(b = -)$ and $(c = +)$. If we delete from $\widehat{P^{t-o}}$ the rules $\{a \leftarrow b,\ b \leftarrow not\ c^o\}$ and assert the fact $\{c^o \leftarrow\}$, denoting the resulting program by P^*, then we have

$$WFM_P(P) = \triangledown \triangle \widehat{P^{t-o}} = \triangledown \triangle \widehat{P^*},$$

as long as the interpretation $\triangle \widehat{P^*}$ is taken with respect to the language of P^{t-o}. That is, the $WFSX_P$ semantics is invariant under these syntactic changes. Notice that if we envisage the semantic changes from the interpretation $\triangle \widehat{P^{t-o}}$ to the interpretation $\triangle \widehat{P^*}$, under the logic Twenty, we verify that (see table 5.1):

- The *Twenty* valuation of a corresponds to the logic value 11 in $\triangle \widehat{P^{t-o}}$, i.e. $(a^o = -,\ a = u,\ \neg a^o = u,\ \neg a = +)$, and to the logic value 6 in $\triangle \widehat{P^*}$, i.e. $(a^o = -,\ a = -,\ \neg a^o = +,\ \neg a = +)$; the *Nine* valuation of a is f in both cases;

- The *Twenty* valuation of b corresponds to the logic value 7 in $\triangle \widehat{P^{t-o}}$, i.e. $(b^o = -,\ b = u,\ \neg b^o = -,\ \neg b = -)$, and to the logic value 1 in $\triangle \widehat{P^*}$, i.e. $(b^o = -,\ b = -,\ \neg b^o = -,\ \neg b = -)$; the *Nine* valuation of a is IV in both cases;

- The *Twenty* valuation of c corresponds to the logic value 18 in $\triangle \widehat{P^{t-o}}$, i.e. $(c^o = u,\ c = +,\ \neg c^o = -,\ \neg c = -)$, and to the logic value 20 in $\triangle \widehat{P^*}$, i.e. $(c^o = +,\ c = +,\ \neg c^o = -,\ \neg c = -)$; the *Nine* valuation of a is t in both cases;

- The *Twenty* valuation of p corresponds to the logic value 12 in both $\triangle\widehat{P^{t-o}}$ and $\triangle\widehat{P^{*}}$, i.e. ($p^o = u$, $p = u$, $\neg p^o = -$, $\neg p = -$); the *Nine* valuation of p is dt.

5.3 Balanced Layered Remainder

The *balanced layered remainder* of an extended normal logic program P, denoted by bP^{t-o}, is a normal logic program obtained by applying the reduction system $\mapsto_{bLWFS}$ (see subsection 5.3.2) to the $t-o$ transformed P^{t-o} of P. The system $\mapsto_{bLWFS}$ results from $\mapsto_{WFS}$ by exchanging the operator of negative reduction $\mapsto_N$, by the operator of *balanced layered negative reduction*, $\mapsto_{bLN}$ (see def. 5.6). The role of the balanced layered remainder in the computation of MH_P models, is the same as the role of the layered remainder in the computation of MH models: it is used to compute the assumable hypotheses set of a program, in view of using loops as choice devices. The definition of balanced layered negative reduction given in this section, is adequate to eliminate the 4-tuple $(l^o, l, \neg l^o, \neg l)$ valuations that produce undefined literals in $WFSX_P$ models (see table 5.1, subsection 5.4.3), which are the valuations 3, 9, 12, 13, 14 of *Twenty*. The remaining *Twenty* valuations correspond to logic *Six*, which constitutes a truth-functional model theory to the MH_P semantics (see subsection 5.4.3).

5.3.1 Balanced Layered Negative Reduction

We could consider to obtain the MH_P models in exactly the same manner as the one used to obtain the MH models, i.e. taking the layered remainder $\overset{o}{P^{t-o}}$ of P^{t-o}, gathering as hypotheses the atoms there appearing default negated and that are not facts, and proceed to compute the minimal hypotheses models, which would then be "$\bigtriangledown$–read" as MH_P models. This path, alluring as it may be, is by no means satisfactory. To see this is the case, consider the following variant on the vacation problem [PP11], represented by the extended normal logic program P

$$\begin{aligned} b &\leftarrow h \\ h &\leftarrow not\ p \\ p &\leftarrow not\ b \\ b &\leftarrow \\ \neg h &\leftarrow \end{aligned}$$

whose $t-o$ transform is P^{t-o}

$$\begin{aligned} b &\leftarrow h & \qquad b^o &\leftarrow h^o, not\ \neg b \\ h &\leftarrow not\ p^o & \qquad h^o &\leftarrow not\ p, not\ \neg h \\ p &\leftarrow not\ b^o & \qquad p^o &\leftarrow not\ b, not\ \neg p \\ b &\leftarrow & \qquad b^o &\leftarrow not\ \neg b \\ \neg h &\leftarrow & \qquad \neg h^o &\leftarrow not\ h. \end{aligned}$$

Now the layered remainder $\overset{o}{P^{t-o}}$ of this program is

$b \leftarrow h$	~~$b^o \leftarrow h^o, not\ \neg b$~~
$h \leftarrow not\ p^o$	~~$h^o \leftarrow not\ p, not\ \neg h$~~
~~$p \leftarrow not\ b^o$~~	$p^o \leftarrow not\ b,$ ~~$not\ \neg p$~~
$b \leftarrow$	$b^o \leftarrow$ ~~$not\ \neg b$~~
$\neg h \leftarrow$	$\neg h^o \leftarrow not\ h.$

The semantics corresponding to a "$\bigtriangledown\triangle$-reading" of this program would not be correctly posed, since we have $p \in (\triangle \overset{o}{P^{t-o}})^-$ and $p^o \in (\triangle \overset{o}{P^{t-o}})^u$, which is not in accordance with corollary 5.8. The problem with this situation is that it dismisses our intentions of using a proper subset of all the $WFSXp$ possible literal valuations as the set of all possible MH_P literal valuations.

Thus an adaptation of the layered negative reduction must be enforced. For this purpose two options seem reasonable (refer to the program $\overset{o}{P^{t-o}}$ in the example above):

(a) If the body of a rule r (resp. r^o) contains a default literal, say $not\ b^o$ (resp. $not\ b$), where b^o (resp. b) is a fact of the program, then eliminate the rule by negative reduction if rule r (resp. r^o) is not in loop through the atom b^o (resp. b) or its co-rule, r^o (resp. r), is not in loop through the atom b (resp. b^o). With this restriction the remainder $P_{(a)}$ becomes:

$h \leftarrow$	
$b \leftarrow$	$b^o \leftarrow$
$\neg h \leftarrow$	

Notice that when rule r is deleted so is rule r^o. Hence corollary 5.8 is respected.

(b) If the body of a rule r (resp. r^o) contains a default literal, say $not\ b^o$ (resp. $not\ b$), where b^o (resp. b) is a fact of the program, then the negative reduction does not take place if the rule r (resp. r^o) is involved in a loop via the atom b^o (resp. b) or its co-rule, r^o (resp. r) , is involved in a loop via the atom b (resp. b^o) . With this restriction the remainder $P_{(b)}$ becomes:

$b \leftarrow h$	
$h \leftarrow not\ p^o$	
$p \leftarrow not\ b^o$	$p^o \leftarrow not\ b$
$b \leftarrow$	$b^o \leftarrow$
$\neg h \leftarrow$	$\neg h^o \leftarrow not\ h$

and the statement of corollary 5.8 is again respected. Our approach will be the one stated in item (b) above. The approach (a) does not consider the abductive hypotheses that are suggested by the undefined literals in $\triangle P^{t-o}$, and we get the single interpretation $\bigtriangledown \triangle P_{(a)} = \langle \{b, \neg h, h\}^+, \{\}^u, \{\neg b, h, \neg h, p, \neg p\}^- \rangle$, which is inconsistent. On the other hand, by considering the option (b) we also get the interpretation, $\triangledown \vartriangle P_{(a)} = \langle \{b, \neg h, p\}^+, \{\}^u, \{\neg b, h, \neg p\}^- \rangle$, by abducting p (see section 5.4.1), which is consistent. We name *balanced layered negative reduction* (see definition 5.6) this $t-o$ version of the *layered negative reduction*. We also call *balanced layered remainder* of P to the program $P_{(b)}$ (see definition 5.8). Let us define the balanced negative reduction.

Definition 5.6. Balanced[3] layered negative reduction. Let P_1 and P_2 be two ground normal logic programs, whose Herbrand bases may contain "o" superscript and non superscript atoms. We say that P_2 results from P_1 by a *balanced layered negative reduction* operation, $P_1 \mapsto_{bLN} P_2$, iff one of the next two cases occur:

1. There is a fact b^o in P_1 and a rule r in P_1 whose body contains the literal *not* b^o, where neither r is involved in a loop through the literal *not* b^o, nor is its co-rule r^o involved in a loop through the literal *not* b, and $P_2 = P_1 \setminus \{r\}$;

2. There is a fact b in P_1 and a rule r^o in P_1 whose body contains the literal *not* b, where neither r^o is involved in a loop through the literal *not* b, nor is its co-rule r involved in a loop through the literal *not* b^o, and $P_2 = P_1 \setminus \{r^o\}$.[4]

5.3.2 The $\mapsto_{bLWFS}$ Reduction System

Definition 5.7. Balanced layered reduction system. The *balanced layered reduction system* is the system $\mapsto_{bLWFS} := \mapsto_P \cup \mapsto_{bLN} \cup \mapsto_S \cup \mapsto_F \cup \mapsto_L$.

Theorem 5.15. Termination and confluence. The system $\mapsto_{bLWFS}$, when applied to finite ground programs, is both terminating and confluent.

To prove this theorem the below defined lemmas 5.16 and 5.17 are needed.

Lemma 5.16. Let Q be a finite ground normal logic program. Let P, S denote respectively, the transformations of positive reduction and success. Then the reduction system $X := \{P, S\}$ is terminating and confluent when applied on Q.[5]

Proof. See appendix C.2.

Lemma 5.17. Let Q be a finite ground normal logic program. Let also P, S, E denote respectively, the operations of positive reduction, success and erasure, where erasure is any transformation that erases a set of rules, and whose preconditions for application are the presence of a certain set of facts or the absence of a certain set of rules, in the program on which it is applied. The rewriting system $X^* := \{P, S, E\}$ is terminating and confluent when applied on Q.

Proof. See appendix C.2.

[3]The expression "balanced" refers to the consideration of pairs of co-rules in this definition.

[4]This operation is weaker than layered negative reduction, meaning that where the former is applicable so is the latter.

[5]The validity of this lemma is an immediate consequence of the termination and confluency of the system $\mapsto_{WFS}$ [BDFZ01]. The proof of the lemma is included here for purposes of completeness of the subjects exposed.

Proof. (of theorem 5.15) Let Q be a finite ground normal logic program. Termination is immediate, due to the finite number of rules and literals of Q and to the fact that all the five operators of the system $\mapsto_{bLWFS}$ reduce the program (i.e., eliminate rules or literals). The confluence comes after lemma 5.17: the system $\mapsto_{bLWFS}$ can be envisaged as a set of $\{P, S, E\}$ operators, where E can refer to any of the operators $\mapsto_{bLN}, \mapsto_F, \mapsto_L$. $\square$

In the following definition we present the balanced layered remainder of an extended normal logic program.

Definition 5.8. Balanced layered remainder. Let P be a finite ground extended normal logic program and P^{t-o} its $t-o$ transformed. We call *balanced layered remainder* of P to the program bP^{t-o} such that $P^{t-o} \mapsto^*_{bLWFS} bP^{t-o}$.[6]

The next proposition ensures that the possible valuations of 4-tuples $(l^o, l, \neg l^o, \neg l)$ in the model $\triangle bP^{t-o}$, belong to the set of 20 possible valuations represented in table 5.1, column *Twenty*, subsection 5.4.3. This confirms that the balanced layered remainder of an extended normal logic program is correctly defined, so that the set of possible MH_P valuations of 4-tuples is a proper subset of the 20 possible valuations set of *Twenty*.

Proposition 5.18. Let P be an extended normal logic program and $M = \triangle bP^{t-o}$. Then the valuation with respect to M of any pair of co-atoms, say b, b^o, of the language of P^{t-o}, agrees with the statement of corollary 5.8.

Proof. The result stated in corollary 5.8 is valid for the well-founded model of the $t-o$ transformed P^{t-o} of an extended normal logic program P. The WFM of a normal logic program can be computed by resorting to the remainder of the program, through the reduction system $\mapsto_{WFS} := \mapsto_P \cup \mapsto_N \cup \mapsto_S \cup \mapsto_F \cup \mapsto_L$. As the balanced layered remainder of P is computed via the system $\mapsto_{bLWFS} := \mapsto_P \cup \mapsto_{bLN} \cup \mapsto_S \cup \mapsto_F \cup \mapsto_L$, we just need to prove that the transformation bLN does not originate any violation of the possible valuations of co-atoms stated in corollary 5.8. That no such violation may occur stems from the very definition of bLN. $\square$

In [Pin11] the author defines a "layered" version of the WFS, the *layered well-founded semantics*, $LWFS$: given a normal logic program P and the corresponding layered remainder, $\overset{o}{P}$, the layered weel-founded model of P is $LWFM(P) = \triangle \overset{o}{P}$, where the model $\triangle \overset{o}{P}$ is computed with respect to the language of P. In the same vein we define the *balanced paraconsistent well-founded semantics*, $bWFSXp$.

Definition 5.9. Balanced paraconsistent well-founded semantics. The *balanced paraconsistent well-founded semantics* $bWFSX_P$ of an extended normal logic program P, is defined by the model $bWFM_P(P) = \bigtriangledown \triangle bP^{t-o}$.

$LWFM(P)$ is generally a more skeptical model than $WFM(P)$, with respect to knowledge ordering. Also $bWFM_P(P)$ is generally a more skeptical model than $WFM_P(P)$, with respect to knowledge ordering. This is stated in the next theorem.

Theorem 5.19. Let Q be an extended normal logic program. Then

$$(bWFM^+_P(Q) \cup not\ bWFM^-_P(Q)) \leq_k (WFM^+_P(Q) \cup not\ WFM^-_P(Q)). \tag{5.3}$$

Proof. See appendix C.2.

[6] $\mapsto^*_{bLWFS}$ means the non deterministic performing of operations of the system $\mapsto_{bLWFS}$ until the resulting program becomes invariant.

5.4 The MHp Semantics

The computation of the MH_P semantics of an extended normal logic program P resembles the computation of the MH semantics of a normal logic program P (see subsection 3.3.2): (i) Compute the $t-o$ transformed P^{t-o} of P (cf. definition 5.3); (ii) Compute the balanced layered remainder bP^{t-o} of P and use it to get the assumable hypotheses set $Hyps(P)$ of P (see definition 5.10); (iii) Non deterministically chose subsets $H \subseteq Hyps(P)$ such that $WFM_P^u(P \cup H) = \emptyset$; (iv) Minimize the corresponding interpretations $WFM_P(P \cup H)$ with respect to nonempty hypotheses sets H and get the MH_P models; also, if $WFM_P^u(P) = \emptyset$, then $WFM_P(P)$ is a MH_P model (cf. definition 5.11).

5.4.1 MH_P Models

The abduction of an objective literal corresponds to the statement of its positivity. In the case of the remainder of a $t-o$ transformed program, an abduction means the assertion in the remainder of a set of co-rules, say $\{b \leftarrow,\ b^o \leftarrow not\ \neg b\}$. The assertion of non superscript facts alone, $b \leftarrow$, may produce an inconsistency in the resulting models, if the corresponding superscript atom has no longer a rule in the program; the assertion of superscript head rules alone, $b^o \leftarrow \neg b$, may produce incorrect programs – see example in subsection 5.3.1. We follow here the definition of assumable hypotheses set for the MH semantics (see definition 3.16), taking also into account the above referred issues.

Definition 5.10. Assumable hypotheses set of a program. Let P be a finite ground extended normal logic program and bP^{t-o} its balanced layered remainder. We say that $Hyps(P) \subseteq \mathcal{H}_P$ is the *assumable hypotheses set* of P iff for all $h \in Hyps(P)$ it is the case that the default literal *not* h^o appears in program bP^{t-o}, and h is not a fact in bP^{t-o}, i.e. $h \in (\triangle bP^{t-o})^u$.[7]

Example 5.3. The balanced layered remainder bP^{t-o} of program P in subsection 5.3.1 is (the eliminated literals and rules are striped out)

$b \leftarrow h$ ~~$b^o \leftarrow h^o, not\ \neg b$~~

$h \leftarrow not\ p^o$ ~~$h^o \leftarrow not\ p, not\ \neg h$~~

$p \leftarrow not\ b^o$ $p^o \leftarrow not\ b,$ ~~$not\ \neg p$~~

$b \leftarrow$ $b^o \leftarrow$ ~~$not\ \neg b$~~

$\neg h \leftarrow$ $\neg h^o \leftarrow not\ h.$

The set of assumable hypotheses of P is $Hyps(P) = \{p\}$, because p is the only objective literal of the language of P such that $p \in (\triangledown \triangle bP^{t-o})^u$, and *not* p^o appears in the program bP^{t-o}.

Definition 5.11. Paraconsistent minimal hypotheses semantics, MH$_\mathbf{P}$. Let P be a finite ground extended normal logic program, bP^{t-o} its balanced layered remainder and $Hyps(P)$ the set of assumable hypotheses of P. Then the *paraconsistent minimal hypotheses* semantics of P, denoted $MHp(P)$, is defined by the paraconsistent minimal hypotheses models M of P, which are computed as follows:

[7]This is equivalent to say that $h \in (\triangledown \triangle bP^{t-o})^u$. Notice that the purpose of computing bP^{t-o} is to find the set of assumable hypotheses of P.

1. $M = WFM_P(P \cup H) = \bigtriangledown WFM(P^{t-o} \cup H \cup \{h^o \leftarrow not\ \neg h : h \in H\})$,
   for all $H \subseteq Hyps(P)$, $H \neq \emptyset$, H is inclusion-minimal and $WFM_P^u(P \cup H) = \emptyset$;
2. $M = WFM_P(P) = \bigtriangledown WFM(P^{t-o})$, where $WFM_P^u(P) = \emptyset$.

The following definition and proposition show that MH_P is a *total paraconsistent models* semantics.

Definition 5.12. Total/Partial Paraconsistent Model. (adapted from [SZ91]) Let SEM be a paraconsistent semantics for extended normal logic programs and M a SEM model of an extended normal logic program P. We say that M is a *total paraconsistent model* iff for every objective literal L of the language of P, either $L \in M$ or $not\ L \in M$. We say that M is a *partial paraconsistent model* iff there is an objective literal L of the language of P, such that $L \notin M$ and $not\ L \notin M$.

Proposition 5.20. The *MHp* semantics is a total paraconsistent models semantics, meaning that for any extended normal logic program P all the MH_P models of P are total paraconsistent models.

Example 5.4. The *MHp* models of the program P in the example of subsection 5.3.1, whose assumable hypotheses set is $Hyps(P) = \{p\}$, are M_1 and M_2:

$$\begin{aligned} H = \emptyset \qquad M_1 &= WFM_P(P) \\ &= \bigtriangledown WFM(P^{t-o}) \\ &= \bigtriangledown \langle \{b, b^o, h, \neg h\}^+, \{\}^u, \{\neg b, \neg b^o, h^o, \neg h^o, p, p^o, \neg p, \neg p^o\}^- \rangle \\ &= \langle \{b, h, \neg h\}^+, \{\}^u, \{\neg b, h, \neg h, p, \neg p\}^- \rangle \end{aligned}$$

$$\begin{aligned} H = \{p\} \qquad M_2 &= WFM_P(P \cup \{p\}) \\ &= \bigtriangledown WFM(P^{t-o} \cup \{p \leftarrow,\ p^o \leftarrow not\ \neg p\}) \\ &= \bigtriangledown \langle \{b, b^o, \neg h, \neg h^o, p, p^o, \}^+, \{\}^u, \{\neg b, \neg b^o, h, h^o, \neg p, \neg p^o\}^- \rangle \\ &= \langle \{b, p, \neg h\}^+, \{\}^u, \{\neg b, h, \neg p\}^- \rangle \end{aligned}$$

M_1 is an inconsistent model whilst M_2 is consistent.

Example 5.5. $\mathbf{MH_P}$ semantics computation. Let P be the program

$$\begin{aligned} k &\leftarrow not\ y, not\ b \\ y &\leftarrow k \\ y &\leftarrow d \\ d &\leftarrow not\ y, b \\ a &\leftarrow b \\ b &\leftarrow not\ c \\ c &\leftarrow not\ a \\ \neg a &\leftarrow \end{aligned}$$

the $t - o$ transform P^{t-o} of it being,

$$\begin{array}{ll}
k \leftarrow not\ y^o, not\ b^o & k^o \leftarrow not\ y, not\ b, not\ \neg k \\
y \leftarrow k & y^o \leftarrow k^o, not\ \neg y \\
y \leftarrow d & y^o \leftarrow d^o, not\ \neg y \\
d \leftarrow not\ y^o, b & d^o \leftarrow not\ y, b^o, not\ \neg d \\
a \leftarrow b & a^o \leftarrow b^o, not\ \neg a \\
b \leftarrow not\ c^o & b^o \leftarrow not\ c, not\ \neg b \\
c \leftarrow not\ a^o & c^o \leftarrow not\ a, not\ \neg c \\
\neg a \leftarrow & \neg a^o \leftarrow not\ a.
\end{array}$$

The balanced layered remainder bP^{t-o} is,

$k \leftarrow not\ y^o,$ ~~$not\ b^o$~~ $\qquad k^o \leftarrow not\ y, not\ b,$ ~~$not\ \neg k$~~

$y \leftarrow k \qquad y^o \leftarrow k^o,$ ~~$not\ \neg y$~~

$y \leftarrow d \qquad$ ~~$y^o \leftarrow d^o, not\ \neg y$~~

$d \leftarrow not\ y^o, b \qquad$ ~~$d^o \leftarrow not\ y, b^o, not\ \neg c$~~

$a \leftarrow b \qquad$ ~~$a^o \leftarrow b^o, not\ \neg a$~~

$b \leftarrow not\ c^o \qquad$ ~~$b^o \leftarrow not\ c, not\ \neg b$~~

$c \leftarrow$ ~~$not\ a^o$~~ $\qquad c^o \leftarrow not\ a,$ ~~$not\ \neg c$~~

$\neg a \leftarrow \qquad \neg a^o \leftarrow not\ a.$

The assumable hypotheses set of P is $\{y\}$. Let us then compute the MH_P models of P:

$H = \emptyset \qquad WFM^u_P(P) = \{k, y\};$ there is no MH_P model for $H = \emptyset$.

$$\begin{aligned}
H = \{y\} \qquad M &= WFM_P(P \cup \{y\}) \\
&= \triangledown WFM(P^{t-o} \cup \{y \leftarrow,\ y^o \leftarrow not\ \neg y\}) \\
&= \triangledown \langle \{\neg a, c, y, y^o\}^+, \{a, \neg a^o, b, c^o\}^u, \\
&\qquad \{a^o, b^o, \neg b, \neg b^o, \neg c, \neg c^o, d, d^o, \neg d, \neg d^o, k, k^o, \neg k, \neg k^o, \neg y, \neg y^o\}^- \rangle \\
&= \langle \{\neg a, c, y\}^+, \{\}^u, \{a, b, \neg b, \neg c, d, \neg d, k, \neg k, \neg y\}^- \rangle
\end{aligned}$$

M is the only MH_P model of P. The model is default consistent, but b, d, k valuations state a contradictory belief.

5.4.2 MH_P Collapses into MH for Normal Logic Programs

The following theorem says that if P is a normal logic program then $MHp(P)$ and $MH(P)$ are the same if we discard from each model $M_p \in MH_P(P)$ the default literals involving explicitly negated literals.

Theorem 5.21. MHp and MH coincide for normal logic programs. Let P be a normal logic program and P^{t-o} the $t-o$ transform of P. Let $MH(P)$ be the set of minimal hypotheses models of P and $MH_P(P)$ the set of paraconsistent minimal hypotheses models of P. Then $M \in MH(P)$ iff there is a model $M_p \in MH_P(P)$, such that $M^+ = M_p^+$ and $M^- = (M_p^- \cap \mathcal{H}_p)$, where $\mathcal{H}_p$ is the Herbrand base of P.

Proof. See appendix C.3.

5.4.3 The Logic *Six*

In [DP98] the authors present a logic, there dubbed *Nine*, which provides a truth-functional model theory for the *WFSXp* semantics, based on Ginsberg's bilattices concept [Gin88]. Here we adopt the ideas that founded the definition of *Nine*, to define a truth-functional logic called *Six*, which in turn constitutes a truth-functional model theory to the *MHp* semantics.

The truth-space of *Nine* comprises nine logic values, here presented together with their meanings: "**t**" and "**f**" are the classical values for truth and falsity; "**I**", is the contradictory truth value; "**II**", is understood as truth with contradictory belief; "**III**", is understood as falsity with contradictory belief; "**IV**", is understood as contradictory belief; "$\perp$", is understood as undefinedness; "**df**", is understood as default falsity; "**dt**", is understood as default truth. Contrary to *Nine*, the truth-space of the logic *Six* does not correspond to a bilattice. It comprises six logic values $\{t, f, I, II, III, IV\}$, which constitute a subset of the truth-space of *Nine*. Table 5.1 represents all possible 4-tuple valuations of a literal $(l^o, l, \neg l^o, \neg l)$ with the corresponding *Twenty*, *Nine* and *Six* logic values. Notice that *Twenty* is the natural logic associated with the well-founded model of the $t-o$ transformed of an extended normal logic program, since there are 20 possible 4-tuple literal valuations as a consequence of corollary 5.8 and the coherence principle. The blank spaces in the column *Six* concern valuations of $(l^o, l, \neg l^o, \neg l)$ that do not occur in a interpretation $\triangle WFM(P^{t-o} \cup H \cup H^o)$, where $\bigtriangledown \triangle\, WFM(P^{t-o} \cup H \cup H^o)$ is a MH_P model of the extended normal logic program P (H^o stands here for $\{h^o \leftarrow not\ \neg h : h \in H\}$, where H is a set of hypotheses).

The relation expressed in the column *Six* of table 5.1, is asserted by the following transformation.

Definition 5.13. τ transformation. (adapted from [DP98]) Let J be a set of literals, default or objective, and $\mathcal{L}_J$ the set of objective literals of the language of J. The *Six* interpretation $\tau(J)$, with underlying language $\mathcal{L}_J$, is the interpretation

$$\forall_{b \in \mathcal{L}_J}\ \tau(b) = \varphi,$$

where $\varphi \in \{f, t, I, II, III, IV\}$ is computed using theorem 5.5 and the correspondence defined in column *Six* of table 5.1 (when taking J as an interpretation, in order to use theorem 5.5, for each literal $b \in \mathcal{L}_J$, any of the literals $b, not\ b, \neg b, not\ \neg b$ not pertaining to J is considered undefined).

The following theorem establishes each *MHp* model of an extended normal logic program P as a *Six* model of the theory P_6 that is obtained straight away from P.

Theorem 5.22. Six models of MH$_\mathbf{P}$. (adapted from [Dam96]) Let P be an extended normal logic program and M a MH_P model of P with hypotheses H. Then $\tau(M)$ is a Six model of P_6, i.e., $\tau(M) \models_6 P_6$. The theory P_6 is obtained from P by replacing in P all program rules $l_0 \leftarrow l_1, \cdots, l_m, not\ l_{m+1}, \cdots not\ l_n$ by the $\mathcal{L}_6$ formula $l_0 \leftarrow l_1 \wedge \cdots \wedge l_m \wedge not\ l_{m+1} \wedge \cdots \wedge not\ l_n$.[8]

[8]For the definitions of the truth-tables of the operators "$\leftarrow$", "*not*", "$\neg$" and "$\wedge$", see [DP98].

Table 5.1: $Twenty - Nine - Six$ valuations correspondence.

l^o	l	$\neg l^0$	$\neg l$	Twenty	Nine	Six
$-$	$-$	$-$	$-$	1	IV	IV
$-$	$-$	$-$	u	2	IV	IV
$-$	$-$	u	u	3	df	
$-$	$-$	$-$	$+$	4	III	III
$-$	$-$	u	$+$	5	f	f
$-$	$-$	$+$	$+$	6	f	f
$-$	u	$-$	$-$	7	IV	IV
$-$	u	$-$	u	8	IV	IV
$-$	u	u	u	9	df	
$-$	u	$-$	$+$	10	III	III
$-$	u	u	$+$	11	f	f
u	u	$-$	$-$	12	dt	
u	u	$-$	u	13	dt	
u	u	u	u	14	$\perp$	
$-$	$+$	$-$	$-$	15	II	II
$-$	$+$	$-$	u	16	II	II
$-$	$+$	$-$	$+$	17	I	I
u	$+$	$-$	$-$	18	t	t
u	$+$	$-$	u	19	t	t
$+$	$+$	$-$	$-$	20	t	t

5.4.4 Properties of the MH_P

The MH_P semantics, being an extension of MH to extended normal logic programs, has the following characterization with respect to existence, relevance and cumulativity.

- It is existential, by theorem 5.23 below.
- It is global to local relevant due to be existential (propositions 4.13, 4.14 are valid also for extended normal logic programs); it is not local to global relevant, since it coincides with MH for normal logic programs. Thus it is not relevant. Meanwhile, MH_P is simple relevant, like MH.
- Due to be coincident with MH for normal logic programs, it is neither cautious monotonic nor cut.

Theorem 5.23. Let P be an extended normal logic program and $Hyps(P)$ the set of assumable hypotheses (possibly empty) of P. Then there is at least a subset $H \subseteq Hyps(P)$ such that

$$WFM_P^u(P \cup H) = \emptyset,$$

which by definition 5.11 means that P has at least one MHp model.

Proof. The set of assumable hypotheses H of an extended normal logic program P, obtained from the program bP^{t-o}, contains the set of assumable hypotheses H' obtained from the program $\widehat{P^{t-o}}$. There is thus a subset $H^* \subseteq H$ such that $WFM_P^u(P \cup H^*) = \emptyset$. Were this not the case, the program $\widehat{P \cup H^*}$ would contain double loops, by proposition 5.13, and the double loops that are not discarded by the loop elimination operation, during the computation of the remainder, must contain some default superscripted literal, say *not* b^o, being b not a fact of the remainder. Hence b would be an assumable hypothesis, meaning that the set H^* above were not as big as it could be. □

5.4.5 Complexity

[9]The theorem below shows that a *brave reasoning* task with MH_P semantics, i.e., finding an MH_P model satisfying some particular set of literals (a query), is in Σ_2^P.

Theorem 5.24. Brave reasoning with MH_P semantics is a Σ_2^P-complete task.

Proof. See appendix C.3.

The theorem below shows that a *cautious reasoning* task with MH_P semantics, i.e. guaranteeing that every MH_P model satisfies some particular set of literals (a query), is a Π_2^P-complete task.

Theorem 5.25. Cautious reasoning with MH_P semantics is a Π_2^P-complete task.

Proof. Cautious reasoning is the complement of brave reasoning, and since the latter is in Σ_2^P the former must necessarily be Π_2^P-complete. □

5.5 Querying an Extended Database

The MH_P semantics may be used to perform reasoning in the following ways, among others (let Q be a query – a conjunction of literals – and P an extended normal logic program in the role of a database).

- **Skeptical consistent reasoning:** Query Q succeeds iff it succeeds for all consistent MH_P models of P;
- **Brave consistent reasoning:** Query Q succeeds iff it succeeds for at least one consistent MH_P model of P;
- **Skeptical paraconsistent reasoning:** Query Q succeeds iff it succeeds for all MH_P models of P, such that none of the objective literals that appear in Q has support on contradiction in any of these models;
- **Brave paraconsistent reasoning:** Query Q succeeds iff it succeeds for at least one MH_P model of P, such that none of the objective literals that appear in Q has support on contradiction in this model;
- **Skeptical liberal reasoning:** Query Q succeeds iff it succeeds for all MH_P models of P;
- **Brave liberal reasoning:** Query Q succeeds iff it succeeds for at least one MH_P model of P.

All the answers above, may come together with additional comments that make clear the *Six* valuation of the conjunction that represents the query.

[9]This subsection follows closely subsection 4.6 of [PP11].

Chapter 6

Dealing with Inconsistency with the MHp Semantics

We present two procedures to extract information from an inconsistent theory, one of them for theory revision and the other for computing the safe literals in the extended kernel of the theory, with respect to the MH_P semantics.

The MH_P semantics has the ability to detect support on contradiction, by generating models that contain L and *not* L for every supported on contradiction objective literal L. That is, MH_P behaves as if attached to each extended normal logic program P, there were an integrity constraint $\bot \leftarrow L, not\ L$ for each objective literal L of the language of P, the violation of which is flagged in the models. In this chapter we want, in a certain way, to extended this detection capability to broader integrity constraint theories. To this purpose, we present two types of actions to perform over inconsistent constrained theories, that may be used separately or in a complementary way. Given an inconsistent constrained theory $P_{ic} = P \cup T_{ic}$, where P is an extended normal logic program and T_{ic} is an integrity constraint theory, (1) we may revise the program in a declarative debugging fashion [PDA93b], as if it were a malfunctioning device, which is done by eliminating some rules and asserting some facts; (2) we may want to figure out the information that is safe in the *extended semantic kernel* of P_{ic}, $kerx_{MH_P}(P) = \bigcap_{M \in MH_P(P)} M$, i.e., the subset of literals of $kerx_{MH_P}(P)$ that are not supported on inconsistency. Both options can be used to perform various types of skeptical and brave query answering.

Through out this chapter, we consider only finite ground constrained theories, meaning that the rules in the theories are ground and in finite number, and each rule, taken as the sequence of symbols that forms it, has a finite length.

The remaining of this chapter goes as follows. In section 6.1, some concepts on the issue of declarative debugging are set forth. An algorithm that revises inconsistent constrained theories, whose outcome is a set of revised theories for each inconsistent constrained theory it takes as input, is defined. In section 6.2 we address the problem of inconsistency propagation detection. An algorithm that computes the subset of *safe* literals in the extended kernel of an inconsistent constrained theory is defined. In section 6.3 we present some ways to answering a query posed to a constrained database, using the revision and inconsistency propagation detection tools here developed. Section 6.4 is for final remarks.

6.1 Revision through Declarative Debugging

A revision of a theory is at stake when all the MHp models of an extended normal logic program P are *inconsistent* with respect to a certain integrity constraint theory T_{ic}, meaning that each model *activates* at least one integrity constraint of T_{ic} (see definition 6.2 below, for active integrity constraint). The path we take here for revising a theory $P_{ic} = P \cup T_{ic}$, consists of deleting some rules of P whose heads are involved in the bodies of *active* integrity constraints (in a way to be defined in the sequel), or asserting facts to revise the CWA valuation of certain literals without a rule (we do not consider asserting facts for any other reason, although this option might be considered and integrated into our revision procedure).[1]

6.1.1 Declarative Debugging

Some terminology is presented below.

Definition 6.1. Integrity constraint theory. Given an extended normal logic program P, an *integrity constraint theory* in the language of P, is a set of nonempty body rules of the type $\bot \leftarrow A, not\ B$, where $\bot$ is a symbol that does not belong to the language of P, A is a conjunction of objective literals and $not\ B$ is a conjunction of default literals, both conjunctions involving only objective literals of the language of P.

Through out this text, every integrity constraint theory is defined using the language of a certain extended normal logic program P, which it goes along with. Also, every integrity constraint theory that goes along with an extended normal logic program, contains implicitly the rules $\bot \leftarrow L, not\ L$ and $\bot \leftarrow L, \neg L$ for every objective literal L of the language of P (for simplicity, these constraints are not represented in the programs throughout this chapter).

Definition 6.2. Active integrity constraint. Let P be an extended normal logic program, T_{ic} an integrity constraint theory in the language of P, and M a model of P under a semantics SEM. An integrity constraint of T_{ic} is said *active* under M iff its body is satisfied by M.

[1]Deleting only the rules whose heads are involved in the bodies of active constraints, in order to deactivate them, has the intention of accepting the maximal number of rules of a program while trying to solve inconsistencies. In a certain way, this option bears a relation with the philosophic *principle of charity*, if we envisage a constrained theory as expressing the convictions of an agent.

Definition 6.3. Model of a constrained theory. Let P_{ic} be a constrained theory and SEM a semantics for extended normal logic programs. A *model* of the theory $P_{ic} = P \cup T_{ic}$ is any SEM model M of P, such that no integrity constraint of T_{ic} is active under M, or M together with the symbol $\perp$ in case some integrity constraint of T_{ic} is active under M.

Definition 6.4. Consistent constrained theory. We say that a constrained theory $P_{ic} = P \cup T_{ic}$ is *consistent* with respect to a semantics SEM, if at least one SEM model of the theory P_{ic} does not contain the symbol $\perp$. In case each SEM model of P contains the symbol $\perp$, the theory is said *inconsistent* with respect to SEM.

The purpose of revising a theory $P_{ic} = P \cup T_{ic}$, is to syntactically alter it in order to obtain a set of *revised* theories, each having at least one model that activates a minimal set of constraints, with respect to the sets of constraints activated by the models of all the revised theories. In order to revise a theory P_{ic}, the below defined *ρ-transformation* is previously performed on the theory.

Definition 6.5. ρ-transformation. (adapted from [PDA93b]) Let $P_{ic} = P \cup T_{ic}$ be a constrained theory. The *ρ-transformed* of P_{ic}, denoted by $\rho(P_{ic})$, is obtained by performing the following modifications on P_{ic}:

1. Add a default literal $not\ del(Head(r))$ to the body of each rule $r \in P$, where the atom $del(Head(r))$ does not belong to the language of P;

2. For each objective literal that appears in P (or that appears in the body of an integrity constraint) and that has no rule in P, say L, include in the transformed program $\rho(P_{ic})$ the rule $L \leftarrow show(L), not\ del(L)$, where the atoms $show(L)$, $del(L)$ do not belong to the language of P.

When computing the MH_P models of $\rho(P_{ic})$, each literal $not\ del(L_1)$ and $not\ show(L_2)$ is considered positive by *CWA*, and any of these assumptions may be revised to avoid inconsistency, in the style of [PDA93a]. Such a revision corresponds, for model computation purposes, to delete every rule whose head is L_1 from the theory P, or to assert the fact L_2 into the theory P (see the algorithm in subsection 6.1.2). We do not consider here the possibility of using more specific arguments with the predicative functions $del()$ and $show()$, although this possibility could be envisaged.

Definition 6.6. Universe of revisables. Let $\rho(P_{ic})$ be the ρ-transformed of a constrained theory P_{ic}. The set of all the atoms involving the predicates $del(), show()$ in $\rho(P_{ic})$, is here referred by *universe of revisables* of the theory P_{ic}, and is denoted by $\mathcal{R}ev(P_{ic})$.[2]

For revision purposes, we may throw into the program $\rho(P_{ic})$ any subset of $\mathcal{R}ev(P_{ic})$, depending on the rules we consider to delete, and on the objective literals without a rule of which we may want to disallow the *CWA* during a revision procedure.

We next define *set of revisables* and *revised program.*

[2]It is implicit that each revision of the theory $\rho(P_{ic})$ corresponds to a revision of the theory P_{ic}. We call "revisables" both to the atoms $del(L_1), show(L_2)$ and to their arguments L_1, L_2.

Definition 6.7. Set of revisables. (adapted from [PDA93a]) A *set of revisables* of a constrained theory P_{ic}, is any subset $\mathcal{R}$ of the universe of revisables $\mathcal{R}ev(P_{ic})$, considered for a revision of P.

By conveniently choosing the subset $\mathcal{R}$, the programmer may select the rules candidate to deletion and the objective literals without a rule for which the *CWA* may not hold.

Definition 6.8. Revised theory. (adapted from [PDA93a]) Given a constrained theory $P_{ic} = P \cup T_{ic}$ and a set $\mathcal{R}$ of revisables of P_{ic}, the *revised theory* [3] with respect to $\mathcal{R}$, is the theory that results from the transformation $\rho(P_{ic}) \cup \mathcal{R} \mapsto_G^* P^*$, where $\mapsto_G$ is the gracious reduction system (see definition 4.3), after eliminating from P^* all facts in $\mathcal{R}$.[4]

To revise a theory P_{ic} under the MH_P semantics, the MH_P models of the transformed theory $\rho(P_{ic})$ must be computed. We thus need to obtain the balanced layered remainder of this theory. We first define the $t-o$ transformed of a constrained theory P_{ic}.

Definition 6.9. Let P_{ic} be a constrained theory. The *$t-o$ transformed* of P_{ic} is the program $P_{ic}^{t-o} = P^{t-o} \cup T_{ic}^{t-o}$, where T_{ic}^{t-o} is obtained by rewriting each T_{ic} rule

$$\bot \leftarrow a_1, \cdots, a_m, not\ b_1, \cdots, not\ b_n, \tag{6.1}$$

as

$$\bot \leftarrow a_1, \cdots, a_m, not\ b_1^o, \cdots, not\ b_n^o, \tag{6.2}$$

the a_i, b_j being objective literals of the language of P.

We next define the ρ-transformation of a theory P_{ic}^{t-o} (an adaptation of definition 6.5).

Definition 6.10. Let $P_{ic} = P \cup T_{ic}$ be a constrained theory. The *ρ-transformed* of P_{ic}^{t-o}, denoted by $\rho(P_{ic}^{t-o})$, is obtained by performing the following modifications on P_{ic}^{t-o}:

1. Add the default literal $not\ del(Head(r))$ to the bodies of each pair of co-rules r, r^o of P^{t-o}, where the atom $del(Head(r))$ does not belong to the language of P;

2. For each objective literal that appears in P (or in the body of an integrity constraint) and that has no rule in P, say L, include in the transformed program $\rho(P_{ic}^{t-o})$ the pair of co-rules $L \leftarrow show(L), not\ del(L)$ and $L^o \leftarrow show(L), not\ del(L), not\ \neg L$, where the atoms $show(L)$, $del(L)$ do not belong to the language of P.

For ease of exposition, we henceforth use the following abbreviations: $Defaults(E)$, is the set of all objective literals that appear default negated in the ground structure E; $ObLiterals(E)$, is the set of all objective literals that appear in the ground structure E. In both cases, E can be a set of rules, a set of literals or a conjunction of literals.

Example 6.1.

$$ObLiterals(\{not \neg a, b, not\ c\}) = \{\neg a, b, c\}$$
$$Defaults(\{not \neg a, b, not\ c\}) = \{\neg a, c\}$$

[3] Some times we may say *revised program* instead of *revised theory*.

[4] Notice that the resulting program involves only literals of the language of P.

The *coherence principle* imposes new dependencies among rules, which lead to an adaptation of the notion of complete rule graph (see definition 2.1) to the case of extended normal logic programs.

Definition 6.11. Complete rule graph. Let P be an extended normal logic program. The *complete rule graph* of P, denoted by $CRG(P)$, is the directed graph whose vertices are the rules of P. Two vertices representing rules r_1 and r_2 define an arc from r_1 to r_2 iff $Head(r_1) \in ObLiterals(Body(r_2))$, or $Head(r_1) = \neg Head(r_2)$.

With interest to the rest of this chapter is the notion of *subprogram relevant to an objective literal* , defined below. This concept is a transcription of the notion of *subprogram relevant to an atom* (see definition 2.5), and uses the notions of *rule depending on a rule* and *objective literal depending on a rule*, which are analogous to related notions already defined for normal logic programs (see definitions 2.2 and 2.4).

Definition 6.12. Subprogram relevant to an objective literal. Let P be an extended normal logic program. We say that a rule $r \in P$ is *relevant* to an objective literal $L \in \mathcal{H}_P$ iff L depends on r. The set of all rules of P relevant to L is represented by $Rel_P(L)$, and is named *subprogram relevant* to L.

Finally, we need also a broader notion of semantic kernel than the one in definition 4.10.

Definition 6.13. Extended semantic kernel. Let P be an extended normal logic program and SEM a total models semantics for extended normal logic programs. We define the *extended semantic kernel* of P with respect to SEM (or simply *extended kernel* of P with respect to SEM), denoted by $kerx_{SEM}(P)$, as the set

$$kerx_{SEM}(P) = \bigcap_{M \in SEM(P)} M, \tag{6.3}$$

where $SEM(P) \neq \emptyset$ and $M = M^+ \cup not\ M^-$.

6.1.2 Revising an Inconsistent Theory

We present now an algorithm to compute the set of revised theories associated with an inconsistent constrained theory P_{ic}, with respect to the MHp semantics. We call it *revision algorithm*. Let us take a look to an informal description of the algorithm, after which a formal definition will be presented.

- For each model M of an inconsistent theory P_{ic}, consider the associated set of active constraints I_M;
- Form the set of revisables, which contains an atom $del(L)$ (resp. $del(\neg L)$) for each objective literal L (resp. $not\ L$) that appears in the body of active constraints in I_M and that has a rule in P, plus the set of all atoms $show()$ in the subprograms relevant to the active constraints;

- Throw each subset of this set of revisables into the theory $\rho(P_{ic})$ and compute the models of the resulting programs. If none of this models activates new constraints, the process is terminated in what concerns model M; otherwise repeat the process with the models involving new active constraints (in a way to be clarified in the formal description of the algorithm);

- By iteratively repeating the previous steps a termination point is reached. By then, consider the set of all models in the leaves of the implicit model searching tree generated by the process, choose the ones that activate smaller sets of constraints and, among these, choose in a Pareto fashion the ones that contain smaller sets of revisables;

- For each model resulting from the previous item, throw into the theory $\rho(P_{ic})$ the set of associated revisables, compute the resulting program, applying the reduction system $\mapsto_G$, and take the subset of rules containing only literals from the language of P_{ic}. Those are the revised theories that the algorithm retrieves.

Revision algorithm

Let $P_{ic} = P \cup T_{ic}$ be an inconsistent constrained theory with respect to the MH_P semantics, where P is a finite ground extended normal logic program and T_{ic} is a finite ground integrity constraint theory in the language of P. Let $P_0 = \rho(P_{ic})$.
Let also

$$S^{show} = \{show(b) : show(b) \in ObLiterals(P_0)\}, \tag{6.4}$$

that is, S^{show} contains all atoms $show()$ that appear in P_0. For an easier reading, we present the first two iteration steps of the algorithm, iteration 1 and iteration 2, after which the general iteration step, iteration n, and the final step, *last step*, will be expounded.

Iteration 1

Compute the finite set of MH_P models of program P_0:[5]

$$\mathcal{M}^1 = \{M_{(1)}, M_{(2)}, \cdots, M_{(n)}\}. \tag{6.5}$$

Suppose that all models are inconsistent. Associated with each model $M_{(i)}$ is the tuple

$$\langle M_{(i)}, IC^{(i)} \rangle, \tag{6.6}$$

where $IC^{(i)}$ is the set of T_{ic} integrity constraints that are active with respect to $M_{(i)}$. For the sake of harmonizing this procedural step with the remaining ones, let $\mathcal{M}^1_* = \mathcal{M}^1$ and $IC_{(i)} = IC^{(i)}$.

Iteration 2

[5]Notice that no literals involving $del()$ or $show()$ are in the assumable hypotheses set of P_0, because such literals are eliminate from the layered remainder by positive reduction (see definition of positive reduction in appendix A).

Consider the following subsets of objective literals involved in the bodies of the constrains in $IC_{(i)}$:

$$All\_IC = ObLiterals\left(Bodies(IC_{(i)})\right), \tag{6.7}$$

is the set of all objective literals involved in the bodies of $IC_{(i)}$ constraints,

$$Def\_IC = Defaults(Bodies(IC_{(i)})), \tag{6.8}$$

is the set of all objective literals that appear default negated in $Bodies(IC_{(i)})$, and

$$Pos\_IC = All\_IC \setminus Def\_IC, \tag{6.9}$$

is the set of all objective literals that appear not default negated in $Bodies(IC_{(i)})$.[6]
Let also

$$S^{-}_{(i)} = \{del(\neg L) : L \in Def\_IC, \neg L \in Heads(P_0)\}, \tag{6.10}$$

$$S^{+}_{(i)} = \{del(L) : L \in Pos\_IC \cap Heads(P_0)\}, \tag{6.11}$$

and let

$$S^{show}_{(i)} = ObLiterals\left(\bigcup_{b\,\in\, All\_IC} Rel_{P_0}(b)\right) \bigcap S^{show} \tag{6.12}$$

be the set of all the $show()$ literals that appear in the union of the subprograms, $\bigcup_{b\,\in\, All\_IC} Rel_{P_0}(b)$, relevant to each objective literal in All_IC.
Do the following.

1. For every model $M_{(i)} \in \mathcal{M}^1$, form the revisables set associated with $M_{(i)}$[7]

$$\mathcal{R}_{(i)} = S^{+}_{(i)} \cup S^{-}_{(i)} \cup S^{show}_{(i)}. \tag{6.13}$$

2. Form the finite set of programs

$$\mathcal{P}^1 = \{\cdots, P_{(i)}, \cdots\}, \tag{6.14}$$

obtained by considering, for all models $M_{(i)} \in \mathcal{M}^1$, the programs

$$P_{(i)} = P_0 \diamond \mathcal{R}_{(i)} \tag{6.15}$$

where the operator $\diamond$ is defined as follows: for every $b \in \mathcal{R}_{(i)}$, add to P_0 the set of rules $\{b \leftarrow pos(b),\ pos(b) \leftarrow not\ neg(b),\ neg(b) \leftarrow not\ pos(b)\}$, where $pos(b), neg(b)$ do not belong to the language of $\rho(P_{ic})$.[8] The models containing the same set of $\mathcal{R}_{(i)}$ elements, represent the MH_P semantics of a certain revised program of P_0 (which may be revised itself in the sequel).

[6]In the above designations All_IC, Def_IC, Pos_IC we have omitted the subindex (i) for simplicty.

[7]The reason for the second member of equality (6.13) is as follows: in order to deactivate a constraint, say $\bot \leftarrow a, not\ b$, we may need to assert $del(a)$ (resp. $del(\neg b)$) to erase each rule r with $Head(r) = a$ (resp. $Head(r) = \neg b$); that is why $S^{+}_{(i)}$ and $S^{-}_{(i)}$ are there; we may also want to change the negative default CWA valuation of some literals with no rule in P; that is why $S^{show}_{(i)}$ is there.

[8]This set of rules acts as a switch, either for adding b to a certain model of $P_{(i)}$, or else for making b false in a certain model of $P_{(i)}$.

3. Compute the MH_P models $M_{(i,j)}$ of each program $P_{(i)} \in \mathcal{P}^1$, and consider the set of all such models, for all programs $P_{(i)} \in \mathcal{P}^1$:

$$\mathcal{M}^2 = \{\cdots, M_{(i,1)}, M_{(i,2)}, \cdots, M_{(i,n_i)}, \cdots\}, \tag{6.16}$$

where each model, say $M_{(i,j)}$, is associated with the tuple

$$\langle M_{(i,j)}, IC^{(i,j)} \rangle, \tag{6.17}$$

being $IC^{(i,j)}$ the set of integrity constraints that are active with respect to $M_{(i,j)}$.

Let

$$\mathcal{M}^2_* = \{M_{(i,j)} \in \mathcal{M}^2 : IC^{(i,j)} \setminus IC_{(i)} \neq \emptyset\}, \tag{6.18}$$

that is, $\mathcal{M}^2_*$ contains all the models $M_{(i,j)}$ in $\mathcal{M}^2$ that activate at least one integrity constraint that is not in the set $IC_{(i)}$. If $\mathcal{M}^2_* = \emptyset$, then the iterative part of the algorithm is over, and the execution continues in the *last step.* If $\mathcal{M}^2_* \neq \emptyset$, then new active constraints are now involved in the revision process, and the execution continues in the step iteration 3, which is a particular case of the general iteration step, iteration n, presented below.

Iteration n

Consider the finite nonempty set of models

$$\mathcal{M}^{n-1}_* = \{\cdots, M_{(s_1,s_2,\cdots,s_{n-1})}, \cdots\}, \tag{6.19}$$

computed at the $(n-1)^{th}$ iteration step, $n-1 \geq 1$, where $s_j \in \mathbb{N}$ and each model, say $M_{(s_1,s_2,\cdots,s_{n-1})}$, is associated with the tuple

$$\langle M_{(s_1,s_2,\cdots,s_{n-1})}, IC^{(s_1,s_2,\cdots,s_{n-1})} \rangle, \tag{6.20}$$

being $IC^{(s_1,s_2,\cdots,s_{n-1})} \setminus IC_{(s_1,s_2,\cdots,s_{n-2})} \neq \emptyset$, and $IC_{(s_1,s_2,\cdots,s_{n-2})} = \emptyset$ if $n-2=0$, where $IC^{(s_1,s_2,\cdots,s_{n-1})}$ is the set of integrity constraints that are active with respect to $M_{(s_1,s_2,\cdots,s_{n-1})}$. Let

$$IC_{(s_1,s_2,\cdots,s_{n-1})} = IC^{(s_1,s_2,\cdots,s_{n-1})} \cup IC_{(s_1,s_2,\cdots,s_{n-2})},$$

that is, $IC_{(s_1,s_2,\cdots,s_{n-1})}$ contains all the constraints activated in the first $n-1$ steps of computation. Analogously to what was donne in iteration step 2, consider the following sets of objective literals:[9]

$$All\_IC = ObLiterals\left(Bodies(IC_{(s_1,s_2,\cdots,s_{n-1})})\right) \tag{6.21}$$

$$Def\_IC = Defaults(Bodies(IC_{(s_1,s_2,\cdots,s_{n-1})})) \tag{6.22}$$

$$Pos\_IC = All\_IC \setminus Def\_IC. \tag{6.23}$$

[9]In the designations All_IC, Def_IC, Pos_IC we have omitted the subindex tuple $(s_1, s_2, \cdots, s_{n-1})$ for simplicty.

Let also

$$S^{-}_{(s_1,s_2,\cdots,s_{n-1})} = \{del(\neg L) : L \in Def\_IC, \neg L \in Heads(P_0)\}, \tag{6.24}$$

$$S^{+}_{(s_1,s_2,\cdots,s_{n-1})} = \{del(L) : L \in Pos\_IC \cap Heads(P_0)\}, \tag{6.25}$$

and let

$$S^{show}_{(s_1,s_2,\cdots,s_{n-1})} = ObLiterals\left(\bigcup_{b \in All\_IC} Rel_{P_0}(b)\right)\bigcap S^{show} \tag{6.26}$$

be the set of all $show()$ literals that appear in the union of the subprograms, $\bigcup_{b \in All\_IC} Rel_{P_0}(b)$, relevant to each objective literal in All_IC.
Do the following.

1. For every model $M_{(s_1,s_2,\cdots,s_{n-1})} \in M^{n-1}_{*}$, form the set of revisables associated with the model,

$$\mathcal{R}_{(s_1,s_2,\cdots,s_{n-1})} = S^{+}_{(s_1,s_2,\cdots,s_{n-1})} \cup S^{-}_{(s_1,s_2,\cdots,s_{n-1})} \cup S^{show}_{(s_1,s_2,\cdots,s_{n-1})}. \tag{6.27}$$

2. Form the finite set of programs

$$\mathcal{P}^{n-1} = \{\cdots, P_{(s_1,s_2,\cdots,s_{n-1})}, \cdots\}, \tag{6.28}$$

obtained by considering, for each model $M_{(s_1,s_2,\cdots,s_{n-1})} \in M^{n-1}_{*}$ and corresponding revisables set, $\mathcal{R}_{(s_1,s_2,\cdots,s_{n-1})}$, the program

$$P_{(s_1,s_2,\cdots,s_{n-1})} = P_0 \diamond \mathcal{R}_{(s_1,s_2,\cdots,s_{n-1})} \tag{6.29}$$

where the meaning of the operator $\diamond$ is analogous to what was stated in iteration step 2.

3. Compute the MH_P models $M_{(s_1,s_2,\cdots,s_n)}$ of each program $P_{(s_1,s_2,\cdots,s_{n-1})} \in \mathcal{P}^{n-1}$, and consider the class of all such models, for all programs $P_{(\alpha_1,\alpha_2,\cdots,\alpha_{n-1})} \in \mathcal{P}^{n-1}$:

$$\mathcal{M}^{n} = \{\cdots, M_{(s_1,s_2,\cdots,s_n)}, \cdots\}, \tag{6.30}$$

where each model, say $M_{(\alpha_1,\alpha_2,\cdots,\alpha_n)}$, is associated with the tuple

$$\langle M_{(\alpha_1,\alpha_2,\cdots,\alpha_n)}, IC^{(\alpha_1,\alpha_2,\cdots,\alpha_n)}\rangle, \tag{6.31}$$

being $IC^{(\alpha_1,\alpha_2,\cdots,\alpha_n)}$ the set of integrity constraints that are active with respect to $M_{(\alpha_1,\alpha_2,\cdots,\alpha_n)}$.

Let

$$\mathcal{M}^{n}_{*} = \{M_{(\alpha_1,\cdots,\alpha_n)} \in \mathcal{M}^{n} : IC^{(\alpha_1,\cdots,\alpha_n)} \setminus IC_{(\alpha_1,\cdots,\alpha_{n-1})} \neq \emptyset\}, \tag{6.32}$$

that is, $\mathcal{M}^{n}_{*}$ contains all the models $M_{(\alpha_1,\cdots,\alpha_n)}$ in $\mathcal{M}^{n}$ that activate at least one integrity constraint that is not in the set $IC_{(\alpha_1,\cdots,\alpha_{n-1})}$. If $\mathcal{M}^{n}_{*} = \emptyset$, then the iterative part of the algorithm is over, and the execution continues in the *last step.* If $\mathcal{M}^{n}_{*} \neq \emptyset$, then new active constraints are now involved in the revision process, and the execution continues in the iteration step $n+1$, which is a particular case of the iteration step n here presented.

Last step

Consider the set of models $\Lambda = \bigcup_{i=1}^{k}(\mathcal{M}^i \setminus \mathcal{M}_*^i)$, where k is the number of iteration steps previously performed. Associated with each model $M_{(\Phi)} \in \Lambda$ is the tuple $\langle M_{(\Phi)}, IC^{(\Phi)}\rangle$ as stated before. It is conceivable a number of preference criteria on the tuples $\langle M_{(\Phi)}, IC^{(\Phi)}\rangle$ and on the programs used to compute them, to select the set of revised programs intended for the theory P_{ic}. As an example of such a preferential heuristics, we are using the following one in this chapter:

1. Consider the models $M_{(\Phi)} \in \Lambda$ with minimal associated $IC^{(\Phi)}$ sets. For a set of models with equal associated IC sets, choose the models containing minimal subsets of literals involving the revisables $show(), del()$. That is, we prefer models with the smallest associated IC sets and, among these, we take those that contain minimal sets of revisables. Let $\mathcal{M}_R$ be the resulting set of models;
2. For each model $M \in \mathcal{M}_R$, take the program P_0 and perform the transformation $P_0 \cup \mathcal{R}_M \mapsto_G^* P_M$, where $\mathcal{R}_M$ is the set of revisables $show(), del()$ contained in M, and $\mapsto_G$ is the gracious reduction system, and then eliminate all the facts in $\mathcal{R}_M$ from program P_M. The set of all programs P_N obtained for all models $N \in \mathcal{M}_R$ after deleting the facts $\mathcal{R}_N$ from P_N, is the set of revised programs generated by this algorithm.

The algorithm just described terminates, as stated by the next proposition.

Proposition 6.1. The *revision algorithm* terminates for every finite ground constrained theory $P_{ic} = P \cup T_{ic}$.

Proof. For the algorithm to terminate, an iterative step n must be reached such that $\mathcal{D}_*^n = \emptyset$. This means that for every model $M_{(\alpha_1,\cdots,\alpha_n)} \in \mathcal{D}^n$, it must be $IC^{(\alpha_1,\cdots,\alpha_n)} \setminus IC_{(\alpha_1,\cdots,\alpha_{n-1})} = \emptyset$. That this is always the case, stems from the fact that all sequences

$$IC_{(\alpha_1)}, IC_{(\alpha_1,\alpha_2)}, \cdots, IC_{(\alpha_1,\alpha_2,\cdots,\alpha_k)}, \cdots, \tag{6.33}$$

are monotonically increasing with respect to set inclusion, and the constrained theory is finite. □

Let us see a couple of examples of application of the revision algorithm.

Example 6.2. Let us revise the inconsistent theory P_{ic},

$$\bot \leftarrow not\ \neg b$$
$$b \leftarrow$$

where $\rho(P_{ic})$ and $\rho(P_{ic}^{t-o})$ are below, respectively in the left and in the center and right columns.

$\rho(P_{ic})$	$\rho(P_{ic}^{t-o})$	
$\bot \leftarrow not\ \neg b$	$\bot \leftarrow not\ \neg b^o$	
$b \leftarrow not\ del(b)$	$b \leftarrow not\ del(b)$	$b^o \leftarrow not\ del(b), not\ \neg b$
$\neg b \leftarrow show(\neg b), not\ del(\neg b)$	$\neg b \leftarrow show(\neg b), not\ del(\neg b)$	$\neg b^o \leftarrow show(\neg b), not\ del(\neg b), not\ b$

We have $P_0 = \rho(P_{ic})$ and $S^{show} = \{show(\neg b)\}$.

Iteration 1

The program $\rho(P_{ic})$ has a single MH_P model $M_{(1)}$, with empty hypotheses set $H_{(1)}$ (in the "$\bigtriangledown$-reading" below, no information on the "o" superscript versions of $\bot, show(), del()$ is considered).

$$\begin{aligned} M_{(1)} &= \bigtriangledown\{b, b^o, not\ \neg b, not\ \neg b^o, not\ show(\neg b), not\ del(b), not\ del(\neg b), \bot\} \\ &= \{b, not\ \neg b, not\ show(\neg b), not\ del(b), not\ del(\neg b), \bot\} \qquad H_{(1)} = \emptyset \\ IC_{(1)} &= IC^{(1)} = \{\bot \leftarrow not\ \neg b\} \\ \mathcal{D}^1 &= \mathcal{D}^1_* = \{M_{(1)}\}. \end{aligned}$$

Notice that $M_{(1)}$ is computed as follows: compute the balanced layered remainder of $\rho(P_{ic}^{t-o})$; consider the hypotheses set $H_{(1)} = \emptyset$; compute the well-founded model of the balanced layered remainder and "$\bigtriangledown$-read" the model $M_{(1)}$, attending to the above referred restrictions. As $\bot \in M_{(1)}$, the theory is inconsistent. A second iteration step must take place.

Iteration 2

$$\begin{aligned} S^-_{(1)} &= \{del(b)\}, \qquad S^+_{(1)} = \emptyset, \qquad S^{show}_{(1)} = \{show(\neg b)\} \\ \mathcal{R}_{(1)} &= \{del(b), show(\neg b)\}. \end{aligned}$$

The program $P^{t-o}_{(1)} = \rho(P^{t-o}_{ic}) \diamond \mathcal{R}_{(1)}$ is

$$\begin{aligned} \bot &\leftarrow not\ \neg b^o & & \\ b &\leftarrow not\ del(b) & b^o &\leftarrow not\ del(b), not\ \neg b \\ \neg b &\leftarrow show(\neg b), not\ del(\neg b) & \neg b^o &\leftarrow show(\neg b), not\ del(\neg b), not\ b \\ del(b) &\leftarrow y & & \\ show(\neg b) &\leftarrow x & & \\ y &\leftarrow not\ y* & & \\ y* &\leftarrow not\ y & & \\ x &\leftarrow not\ x* & & \\ x* &\leftarrow not\ x & & \end{aligned}$$

where x, x^*, y, y^* stand respectively for $pos(show(\neg b)), neg(show(\neg b)), pos(del(b))$, $neg(del(b))$. Notice that the even loops involving x, x^*, y, y^* act as switches. The MH_P models of $P^{t-o}_{(1)}$ are (we discard in the models literals involving x, x^*, y, y^*; the sets $R_{(i,j)}$

contain the true literals involving revisables, in each model):[10]

$$
\begin{aligned}
M_{(1,1)} &= \triangledown\{not\ b, not\ b^o, \neg b, \neg b^o, del(b), show(\neg b)\} = \{not\ b, \neg b\}\\
H_{(1,1)} &= \{x, y\}, \qquad R_{(1,1)} = \{del(b), show(\neg b)\}\\
IC^{(1,1)} &= \emptyset \qquad IC^{(1,1)} \setminus IC_{(1)} = \emptyset\\
M_{(1,2)} &= \triangledown\{b, not\ b^o, \neg b, not\ \neg b^o, not\ del(b), show(\neg b), \bot\} = \{b, not\ b, \neg b, not\ \neg b, \bot\}\\
H_{(1,2)} &= \{x, y^*\}, \qquad R_{(1,2)} = \{not\ del(b), show(\neg b)\}\\
IC^{(1,2)} &= \{\bot \leftarrow not\ \neg b, \bot \leftarrow b, not\ b, \bot \leftarrow \neg b, not\ \neg b\}\\
& \qquad IC^{(1,2)} \setminus IC_{(1)} = \{\bot \leftarrow b, not\ b, \bot \leftarrow \neg b, not\ \neg b\}\\
M_{(1,3)} &= \triangledown\{b, b^o, not\ \neg b, not\ \neg b^o, not\ del(b), not\ show(\neg b), \bot\} = \{b, not\ \neg b, \bot\}\\
H_{(1,3)} &= \{x^*, y^*\}, \qquad R_{(1,3)} = \{not\ del(b), not\ show(\neg b)\}\\
IC^{(1,3)} &= \{\bot \leftarrow not\ \neg b\} \qquad IC^{(1,3)} \setminus IC_{(1)} = \emptyset\\
M_{(1,4)} &= \triangledown\{not\ b, not\ b^o, not\ \neg b, not\ \neg b^o, del(b), not\ show(\neg b), \bot\} = \{not\ b, not\ \neg b, \bot\}\\
H_{(1,4)} &= \{x^*, y\}, \qquad R_{(1,4)} = \{del(b), not\ show(\neg b)\}\\
IC_{(1,4)} &= \{\bot \leftarrow not\ \neg b\} \qquad IC^{(1,4)} \setminus IC_{(1)} = \emptyset\\
\mathcal{D}^2 &= \{M_{(1,1)}, M_{(1,2)}, M_{(1,3)}, M_{(1,4)}\} \quad \mathcal{D}^2_* = \{M_{(1,2)}\}
\end{aligned}
$$

As $\mathcal{D}^2_* \neq \emptyset$, a third iteration step must take place.

Iteration 3

$$
\begin{aligned}
S^-_{(1,2)} &= S^+_{(1,2)} = \{del(b), del(\neg b)\}, \qquad S^{show}_{(1,2)} = \{show(\neg b)\}\\
\mathcal{R}_{(1,2)} &= \{del(b), del(\neg b), show(\neg b)\}.
\end{aligned}
$$

The $t-o$ version of program $P_{(1,2)} = \rho(P_0) \diamond \mathcal{R}_{(1,2)}$ is (where $(\cdots)$ stands for the switching rules for choosing among $\mathcal{R}_{(1,2)}$ subsets):

$$
\begin{array}{rlrl}
\bot & \leftarrow not\ \neg b^o & & \\
b & \leftarrow not\ del(b) & b^o & \leftarrow not\ del(b), not\ \neg b\\
\neg b & \leftarrow show(\neg b), not\ del(\neg b) & \neg b^o & \leftarrow show(\neg b), not\ del(\neg b), not\ b\\
del(b) & \leftarrow x & & \\
del(\neg b) & \leftarrow y & & \\
show(\neg b) & \leftarrow z & & \\
 & (\cdots) & &
\end{array}
$$

where x, y, z are abbreviations for respectively $pos(del(b)), pos(del(\neg b)), pos(show(\neg b))$. Now adding to the program the revisable $del(\neg b)$ is of no help for deactivating the constraint $\bot \leftarrow not\ \neg b$. The consistent models obtained in $\mathcal{M}^3$ have equal or bigger sets of revisables than the models of $\mathcal{M}^2$. Hence they are not considered for computing revised theories. As for no model $M_{(1,2,\gamma)} \in \mathcal{M}^3$ would we have $IC^{(1,2,\gamma)} \setminus IC_{(1,2)} \neq \emptyset$, it is the case that $\mathcal{D}^3_* = \emptyset$. This terminates the iterative steps and thus the *last step* must be performed.

[10]Consider in the models $M_{(i,j)}$ only the literals of the language of P, plus eventually the symbol $\bot$.

Last step

We have

$$\mathcal{D}^1 \setminus \mathcal{D}^1_* = \emptyset$$
$$\mathcal{D}^2 \setminus \mathcal{D}^2_* = \{M_{(1,1)}, M_{(1,3)}, M_{(1,4)}\}$$

and $\mathcal{D}^3 \setminus \mathcal{D}^3_*$ may be neglected. Hence

$$\mathcal{D} = \{M_{(1,1)}, M_{(1,3)}, M_{(1,4)}\}.$$

As $M_{(1,1)}$ is the model with a smaller associated IC set, $IC_{(1,1)} = \emptyset$, the set of revised programs is obtained as follows: take program $P_{(0)}$, add to it the revisables set $\{del(b), show(\neg b)\}$, apply the $\mapsto_G$ reduction system, and discard the facts involving objective literals that do not belong to the language of P and the sets of switching rules. We get the single revised theory

$$\bot \leftarrow not\ \neg b$$
$$\neg b \leftarrow$$

Example 6.3. Consider the constrained theory P_{ic}. Let us see that the theory is inconsistent, and apply on it the revision algorithm.

$$\bot \leftarrow not\ a$$
$$a \leftarrow b$$
$$b \leftarrow not\ a, not\ x$$
$$x \leftarrow$$

Iteration 1

The "$\bigtriangledown$-reading" below has the restrictions stated in example 6.2 (we omit the transformed $\rho(P_{ic}^{t-o})$).

$$\begin{aligned} M_{(1)} = \bigtriangledown\{&not\ a, not\ a^o, not\ \neg a, not\ \neg a^o, not\ b, not\ b^o, not\ \neg b, not\ \neg b^o, x, x^o, not\ \neg x,\\ &not\ \neg x^o, not\ del(a), not\ show(\neg a), not\ show(\neg b), not\ show(\neg x), \bot\} =\\ = \{&not\ a, not\ \neg a, not\ b, not \neg b, x, not\ \neg x, not\ show(\neg a), not\ show(\neg b),\\ &not\ show(\neg x), \bot\} \qquad H_{(1)} = \emptyset\\ IC_{(1)} = IC^{(1)} &= \{\bot \leftarrow not\ a\}\\ \mathcal{D}^1 = \mathcal{D}^1_* &= \{M_{(1)}\}. \end{aligned}$$

As $\bot \in M_{(1)}$ the theory is inconsistent, and thus a second iteration step must take place.

Iteration 2

$$S^-_{(1)} = \emptyset, \qquad S^+_{(1)} = \emptyset, \qquad S^{show}_{(1)} = \emptyset$$
$$\mathcal{R}_{(1)} = \emptyset$$

We have then $P_{(1)} = P_0 \diamond \mathcal{R}_{(1)} = P_0$. Hence theory P_{ic} coincides with its only revision.

6.2 Inconsistency Propagation Detection

Let $P_{ic} = P \cup T_{ic}$ be an inconsistent constrained theory with respect to a total models semantics SEM. Let $kerx_{SEM}(P) = \bigcap\limits_{M \in SEM(P)} M$, and $L \in kerx_{SEM}(P)$, where L is either an objective or a default literal. A question arises: *how can we trust* L, when considering the semantics of P_{ic}? That is, having the theory inconsistent nontrivial models, is the valuation of L a cause of inconsistency, a consequence of inconsistency, or is the definition of L in turn independent from the valuations of the literals that "instigate" inconsistency?

In this section we put forward a procedure that classifies as *suspicious* all the $kerx_{SEM}(P)$ objective literals involved in the bodies of integrity constraints active under some $SEM(P)$ model, or objective literals that depend on these in a way to be clarified in the sequel. The valuations of $kerx_{SEM}(P)$ suspicious literals are not to be trusted. The literals in $kerx_{SEM}(P)$ that are not suspicious, are said to be *safe*.[11]

6.2.1 Motivation

We use an example to clarify our intentions.

Example 6.4. Consider the theory $P_{ic} = P \cup T_{ic}$

$$\begin{aligned} \bot &\leftarrow a, b \\ v &\leftarrow a \\ u &\leftarrow not\ c \\ a &\leftarrow c \\ b &\leftarrow c \\ c &\leftarrow \end{aligned}$$

that admits the single MH_P model $M = \{a, b, c, not\ u, v, \bot\}$ with hypotheses set $H_M = \emptyset$, and hence $kerx_{MH_P}(P) = \{a, b, c, not\ u, v\}$. We do not want to trust the valuations of a, b since both atoms appear in the body of the integrity constraint $\bot \leftarrow a, b$ that is active under M. If the valuation of a is not to be trusted, so is the valuation of v since it depends on the valuation of a. Thus our procedure will identify the literals $\{a, b, v\}$ as suspicious and the literals $\{c, not\ u\}$ as safe.

6.2.2 Semantic Support on Inconsistency

To formalize a process for detecting suspicious and safe literals in the extended kernel of a program, under semantics MH_P, we set forth the notions of *syntactic expansion* of a set of literals and *semantic support on inconsistency* of a literal.

Definition 6.14. Syntactic expansion of a set of literals. Let P be an extended normal logic program, S a set of objective literals of the language of P and SEM a

[11] The designations "suspicious" and "safe" are here adopted after the paper [Sak92], although the author uses these terms to characterize only the semantic dependence on contradiction with respect to explicit negation.

semantics of total models for extended normal logic programs. We call *syntactic expansion* of S in P with respect to SEM, denoted by $EXP_{SEM}(S,P)$, the set of programs obtained as follows:

1. If $S = \emptyset$, then $EXP_{SEM}(S,P) = P$.
2. If $S \neq \emptyset$, then for every element $s \in S$ non deterministically replace in P the set of rules defining s either by the fact $s \leftarrow$, or else by the empty set; if S has k distinct elements, then $EXP_{SEM}(S,P)$ is the set of the 2^k resulting programs.

We say that each objective literal $s \in S$ is *expanded* (in $EXP_{SEM}(S,P)$).

We next define *semantic support on inconsistency of a literal.* The definition is a brief description of an algorithm that constructs the set of all safe literals of the extended kernel of a program. A detailed description of this algorithm is given in subsection 6.2.3.

Definition 6.15. Semantic support on inconsistency[12]**.** Let $P_{ic} = P \cup T_{ic}$ be an inconsistent constrained theory under a total models semantics SEM. Let $L \in kerx_{SEM}(P)$ be a literal of the language of P. Let every SEM model $M_{(i)}$ of P_{ic} be associated with a tuple $\langle M_{(i)}, IC_{(i)} \rangle$, where $IC_{(i)}$ is the subset of T_{ic} integrity constraints that are active under $M_{(i)}$. Let $S_{(i)}$ be the collection of all objective literals involved in the bodies of the integrity constraints in $IC_{(i)}$. Consider the set $\mathcal{D}$ of all models computed as follows.

1. For every SEM model tuple $\langle M_{(i)}, IC_{(i)} \rangle$, obtain the syntactic expansion of $S_{(i)}$, $EXP_{SEM}(S_{(i)}, P)$. Let $\{P(\alpha) : \alpha \in \{1, 2, \cdots, s\},\ s \in \mathbb{N}\}$, be the set of all resulting programs.
2. Repeat step 1 for every SEM model tuple $\langle M_{(\alpha,j)}, IC_{(\alpha,j)} \rangle$ associated with each model $M_{(\alpha,j)}$, for all programs $P(\alpha)$ resulting from the previous step, as long as the set $S_{(\alpha,j)}$ of all objective literals involved in the bodies of active constraints under $M_{(\alpha,j)}$ contains objective literals that were not yet expanded; expand only the subset of $S_{(\alpha,j)}$ containing those literals, for each $S_{(\alpha,j)}$. Keep the process going on iteratively, until there results no model tuple $\langle M_{(\Phi)}, IC_{(\Phi)} \rangle$ such that $S_{(\Phi)}$ contains not yet expanded objective literals.

Let $\mathcal{D}$ be the subset of models computed in this process, such that $M_{(\beta)} \in \mathcal{D}$ iff $S_{(\beta)}$ does not contain not yet expanded literals. We say that $L \in kerx_{SEM}(P)$ has *semantic support on inconsistency* iff $L \notin \bigcap_{M \in \mathcal{D}} M$; in that case, we say that L is a *suspicious literal* of $kerx_{SEM}(P)$ with respect to the constraint theory T_{ic}. Otherwise, if $L \in \bigcap_{M \in \mathcal{D}} M$, we say that L is a *safe literal* of $kerx_{SEM}(P)$ with respect to the constraint theory T_{ic}.

We will see in proposition 6.2 that this definition corresponds to a terminating process. The rationale of the definition is as follows. If S_M is the set of objective literals appearing in the bodies of the active integrity constraints under model M, then consider the expansion of S_M to detect the literals of $kerx_{SEM}(P)$ whose semantics change in some of the resulting programs, thus revealing their semantic dependence on S_M. If any new integrity

[12]This definition is an extension of the definition of *literal depending on contradiction* in [DP97], where the authors refer to a semantic dependence on contradiction with respect to explicit negation.

constraint is activated under some meanwhile computed model, then the objective literals involved in its body must also be expanded in order to spot literals of the extended kernel that depend on them.

Taking into consideration all the possible valuations of the atoms of a set S_M associated to a certain SEM model M, as a form of assessing the semantic dependence of the literals in $kerx_{SEM}(P)$ with respect to S_M, may be envisaged as embodying a skeptical interpretation of semantic dependence.

Using the notion of semantic support on inconsistency, we may now give an answer to the question formulated in the introduction of section 6.2: how can we trust the literals in $kerx_{SEM}(P)$, when considering a constrained theory P_{ic}? The answer we propose is: we do not trust the literals in $kerx_{SEM}(P)$ that are suspicious literals; the only literals to be trusted are the safe literals of $kerx_{SEM}(P)$.

6.2.3 Computing the Safe Literals in the Extended Semantic Kernel

We present now a more detailed version of the algorithm for computing the set of safe literals of $kerx_{SEM}(P)$ outlined in definition 6.15, adapted to the MH_P semantics. We call it the *safeness algorithm*.

Safeness algorithm

Let $P_{ic} = P \cup T_{ic}$ be an inconsistent constrained theory, where P and T_{ic} are finite and ground. For an easier reading, we present the first two iteration steps of the algorithm, iteration 1 and iteration 2, after which the general iteration step, iteration n, and the *last step* will be expounded.

Iteration 1

Let

$$\mathcal{D}^1 = \{M_{(1)}, M_{(2)}, \cdots, M_{(n)}\}, \tag{6.34}$$

be the set of MH_P models of P_{ic} (all models are inconsistent). Associated with each model $M_{(i)}$ is the tuple

$$\langle M_{(i)}, IC_{(i)} \rangle, \tag{6.35}$$

where $IC_{(i)}$ is the set of T_{ic} integrity constraints that are active with respect to $M_{(i)}$.
Let

$$S^{(i)} = ObLiterals(Bodies(IC_{(i)})). \tag{6.36}$$

For the sake of harmonizing this procedural step with the remaining ones, let $\mathcal{D}^1_* = \mathcal{D}^1$ and $S_{(i)} = S^{(i)}$.

Iteration 2

For each model $M_{(i)} \in \mathcal{D}^1_*$, do the following transformation on P_{ic}: for each objective literal $L \in S_{(i)}$ (1) erase all the rules with head equal to L, and (2) add to the program the set of rules $\{L \leftarrow \ pos(L)\ ,\ pos(L) \leftarrow not\ neg(L)\ ,\ neg(L) \leftarrow not\ pos(L)\}$, where $pos(L)$ and $neg(L)$ are atoms that do not belong to the language of P.[13] Let $P_{(i)}$ be the resulting program (one such program per model).
Compute the MH_P models of $P_{(i)}$ for all $i \in \{1, 2, \cdots, n\}$.[14] Let them be

$$\mathcal{D}^2 = \{\cdots, M_{(i,1)}, M_{(i,2)}, \cdots, M_{(i,n_i)}, \cdots\}. \tag{6.37}$$

Associated with each model $M_{(i,j)}$ is the tuple

$$\langle M_{(i,j)}, IC_{(i,j)} \rangle, \tag{6.38}$$

where $IC_{(i,j)}$ is the set of T_{ic} integrity constraints that are active with respect to $M_{(i,j)}$. Associated with each model $M_{(i,j)}$ is also the set

$$S_{(i,j)} = S^{(i,j)} \setminus S_{(i)}, \tag{6.39}$$

where $S^{(i,j)} = ObLiterals(Bodies(IC_{(i,j)}))$[15].
Let $\mathcal{D}^2_* \subseteq \mathcal{D}^2$ be such that $M_{(i,j)} \in \mathcal{D}^2_*$ iff $S_{(i,j)} \neq \emptyset$. If $\mathcal{D}^2_* = \emptyset$, then there are no more objective literals to expand in any model in $\mathcal{D}^2$, which finishes the iterative part of the algorithm – go to *last step*. If $\mathcal{D}^2_* \neq \emptyset$, then new active constraints involving not yet expanded objective literals are now in the process, and the algorithm execution continues in iteration step 3, which is a particular case of the iteration step n defined below.

Iteration n

The general step of the algorithm is as follows.
Let $\mathcal{D}^{n-1}$ be the set of models computed in the $(n-1)^{th}$ iteration step of the process, $n - 1 \geq 1$. Let $\mathcal{D}^{n-1}_* \subseteq \mathcal{D}^{n-1}$ be such that for each $M_{(s^i_0, s^i_1, \cdots, s^i_{n-2})} \in \mathcal{D}^{n-1}$,

$$M_{(s^i_0, s^i_1, \cdots, s^i_{n-2})} \in \mathcal{D}^{n-1}_* \text{ iff } S_{(s^i_0, s^i_2, \cdots, s^i_{n-2})} \neq \emptyset, \tag{6.40}$$

where $i \in \mathbb{N}$, $s^i_j \in \mathbb{N}$, and

$$S_{(s^i_0, s^i_1, \cdots, s^i_{n-2})} = \begin{cases} S_{(s^i_0)}, & n-1 = 1, \\ S^{(s^i_0, s^i_1, \cdots, s^i_{n-2})} \setminus \bigcup_{m=0}^{m=n-3} S_{(s^i_0, s^i_1, \cdots, s^i_m)}, & n-1 > 1. \end{cases} \tag{6.41}$$

being $S^{(s^i_0, s^i_1, \cdots, s^i_{n-2})}$ the set of atoms that appear in the bodies of the active T_{ic} integrity constraints under the model $M_{(s^i_0, s^i_2, \cdots, s^i_{n-2})}$, and $\bigcup_{m=0}^{m=n-3} S_{(s^i_0, s^i_1, \cdots, s^i_m)}$ the set of all already expanded objective literals in the path of the algorithm execution that took to $M_{(s^i_0, s^i_2, \cdots, s^i_{n-2})}$.

[13]This set of rules works as a switch, that allows the inclusion of either L or *not* L in each MH_P model of the resulting program. This permits to simulate the syntactic expansion of L in P_{ic} with respect to MH_P.

[14]The hypotheses of each model may contain both objective literals of the language of P, and literals of the types $neg(L), pos(L)$.

[15]Notice that $S_{(i,j)}$ represents the objective literals in the bodies of constraints active under $M_{(i,j)}$, that were not yet expanded.

1. For each model $M_{(s^i_0,s^i_2,\cdots,s^i_{n-2})} \in \mathcal{D}^{n-1}_*$ do the following. Let $P_{(s^i_0,s^i_1,\cdots,s^i_{n-3})}$ be the program that has $M_{(s^i_0,s^i_2,\cdots,s^i_{n-2})}$ as a MH_P model (if $n-1=1$ then $P_{(s^i_0,s^i_2,\cdots,s^i_{n-3})} = P_{ic}$). Form the program $P_{(s^i_0,s^i_1,\cdots,s^i_{n-3})} \cup X_{(s^i_0,s^i_2,\cdots,s^i_{n-2})}$, where $X_{(s^i_0,s^i_2,\cdots,s^i_{n-2})}$ is the set of atoms formed with the predicate names $neg(), pos()$, that appear in $M_{(s^i_0,s^i_1,\cdots,s^i_{n-2})}$, and transform it in the following way:[16]for each objective literal L in $S_{(s^i_0,s^i_1,\cdots,s^i_{n-2})}$, erase all rules with head equal to L, and add to the program the set of rules $\{L \leftarrow pos(L)\ ,\ pos(L) \leftarrow not\ neg(L)\ ,\ neg(L) \leftarrow not\ pos(L)\}$, where $pos(L)$ and $neg(L)$ are atoms that do not belong to the languages of the programs $P_{ic}, P_{(s^i_0)}, P_{(s^i_0,s^i_1)}, \cdots, P_{(s^i_0,s^i_1,\cdots,s^i_{n-3})}$; for each positive literal $neg(L)$ (resp. $pos(L)$) in $X_{(s^i_0,s^i_2,\cdots,s^i_{n-2})}$, erase all the switching rules involving $neg(L)$ (resp. $pos(L)$), and erase the rule $L \leftarrow pos(L)$ (resp. replace the rule $L \leftarrow pos(L)$ by the fact $L \leftarrow$). Let $P_{(s^i_0,s^i_1,\cdots,s^i_{n-2})}$ be the resulting program.

2. Compute all the MH_P models of all programs $P_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-2})}$ that result from the previous item. Let the set of this models be
$$\mathcal{D}^n = \{\cdots, M_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-1})}, \cdots\}. \tag{6.42}$$
Associated with each model $M_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-1})}$ is the tuple
$$\langle M_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-1})}, IC_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-1})}\rangle, \tag{6.43}$$
where $IC_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-1})}$ is the set of integrity constraints that are active with respect to $M_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-1})}$.
Associated with each model $M_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-1})}$ is also the set
$$S_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-1})}, \tag{6.44}$$
computed as shown in formula (6.41).
Let $\mathcal{D}^n_* \subseteq \mathcal{D}^n$ be such that $M_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-1})} \in \mathcal{D}^n_*$ iff $S_{(\alpha^j_0,\alpha^j_1,\cdots,\alpha^j_{n-1})} \neq \emptyset$. If $\mathcal{D}^n_* = \emptyset$, then there are no more objective literals to expand in any model in $\mathcal{D}^n$, which finishes the iterative part of the algorithm – go to *last step*. If $\mathcal{D}^n_* \neq \emptyset$, then new active constraints involving not yet expanded objective literals are now in the process, and the algorithm execution continues in iteration step $n+1$, which is a particular case of the iteration step n here defined.

Last step

Consider that k is the number of iteration steps previously performed. Then the class
$$\mathcal{D} = \bigcup_{i=1}^{k} (\mathcal{D}^i \setminus \mathcal{D}^i_*), \tag{6.45}$$

[16]The already expanded literals are the arguments of the predicative functions $neg(), pos()$ pertaining to $X_{(s^i_0,s^i_2,\cdots,s^i_{n-2})}$.

contains all the models computed in the process, such that $M_\phi \in \mathcal{D}$ iff $S_\phi = \emptyset$. Thus for every model $M_\phi \in \mathcal{D}$ all the atoms involved in the bodies of active integrity constraints under M_ϕ, are already expanded. If a literal $L \in kerx_{MHp}(P)$ is such that $L \notin \bigcap\limits_{M \in \mathcal{D}} M$, then L is a *suspicious literal* of $kerx_{MHp}(P)$. On the other side, if a literal $L \in kerx_{MHp}(P)$ is such that $L \in \bigcap\limits_{M \in \mathcal{D}} M$, then L is a *safe literal* of $kerx_{MHp}(P)$.

Proposition 6.2. The *safeness algorithm* terminates for every inconsistent finite ground theory $P_{ic} = P \cup T_{ic}$.

Proof. For the algorithm to terminate, an iterative step k must exist, such that $\mathcal{D}^k_* = \emptyset$. This only occurs if for all models $M_{(\alpha_1,\cdots,\alpha_k)} \in \mathcal{D}^k$ we have $S_{(\alpha_1,\cdots,\alpha_k)} = \emptyset$. As all the set sequences

$$S_{(\alpha_1)}, S_{(\alpha_1)} \cup S_{(\alpha_1,\alpha_2)}, \cdots, \bigcup_{m=1}^{m=k} S_{(\alpha_1,\alpha_2,\cdots,\alpha_m)}, \cdots, \tag{6.46}$$

are monotonically increasing with respect to set inclusion, and the constrained theory is finite, the existence of such a step k is guaranteed. □

Example 6.5. Let us compute the set of safe literals in the extended kernel of the theory in example 6.4, with respect to the MH_P semantics.

Iteration 1

The theory has a single MH_P model, $M_{(1)}$, with empty hypotheses set $H_{(1)}$.

$$\begin{aligned} M_{(1)} &= \{a, b, c, not\ u, v, \bot\} \\ H_{(1)} &= \emptyset \\ IC_{(1)} &= \{\bot \leftarrow a, b\} \\ S^{(1)} &= \{a, b\} = S_{(1)} \\ ker_{MH_P}(P) &= \{a, b, c, not\ u, v\} \\ \mathcal{D}^1 &= \mathcal{D}^1_* = \{M_{(1)}\}. \end{aligned}$$

Iteration 2

Form the program $P_{(1)}$, whose models (considering the restriction of each model to he language of P_{ic}) coincide with the union of the sets of models of each program in $EXP_{MH_P}(S_{(1)}, P_{ic})$.

$$\begin{aligned}
&P_{(1)}\\
\bot &\leftarrow a, b\\
v &\leftarrow a\\
u &\leftarrow not\ c\\
c &\leftarrow\\
a &\leftarrow pos(a)\\
b &\leftarrow pos(b)\\
pos(a) &\leftarrow not\ neg(a)\\
neg(a) &\leftarrow not\ pos(a)\\
pos(b) &\leftarrow not\ neg(b)\\
neg(b) &\leftarrow not\ pos(b)
\end{aligned}$$

The MH_P models $M_{(i,j)}$ of $P_{(1)}$, together with its hypotheses sets $H_{(i,j)}$ and also the sets $S_{(i,j)}$ of not yet expanded objective literals, are as follows (we omit the computation of $P_{(1)}^{t-o}$; we also omit the default negations involving explicitly negated literals of the language of P.).

$$\begin{aligned}
M_{(1,1)} &= \{a, b, c, not\ u, v, not\ neg(a), not\ neg(b), pos(a), pos(b), \bot\}\\
H_{(1,1)} &= \{pos(a), pos(b)\}\\
S^{(1,1)} &= \{a, b\} \quad S_{(1,1)} = S^{(1,1)} \setminus S_{(1)} = \{a, b\} \setminus \{a, b\} = \emptyset\\
M_{(1,2)} &= \{a, not\ b, c, not\ u, v, not\ neg(a), neg(b), pos(a), not\ pos(b)\}\\
H_{(1,2)} &= \{pos(a), neg(b)\}\\
S^{(1,2)} &= \emptyset \quad S_{(1,2)} = S^{(1,2)} \setminus S_{(1)} = \emptyset\\
M_{(1,3)} &= \{not\ a, b, c, not\ u, not\ v, neg(a), not\ neg(b), not\ pos(a), pos(b)\}\\
H_{(3)} &= \{neg(a), pos(b)\}\\
S^{(1,3)} &= \emptyset \quad S_{(1,3)} = S^{(1,3)} \setminus S_{(1)} = \emptyset\\
M_{(1,4)} &= \{not\ a, not\ b, c, not\ u, not\ v, neg(a), neg(b), not\ pos(a), not\ pos(b)\}\\
H_{(1,4)} &= \{neg(a), neg(b)\}\\
S^{(1,4)} &= \emptyset \quad S_{(1,4)} = S^{(1,4)} \setminus S_{(1)} = \emptyset\\
\mathcal{D}^2 &= \{M_{(1,1)}, M_{(1,2)}, M_{(1,3)}, M_{(1,4)}\} \quad \mathcal{D}^2_* = \emptyset
\end{aligned}$$

As all $S_{(i,j)}$ are empty sets, we have $\mathcal{D}^2_* = \emptyset$ and the iterative part of the process is thus finished. The process continues in the *last step*.

Last step

Since

$$\begin{aligned}
\mathcal{D}^1 \setminus \mathcal{D}^1_* &= \emptyset\\
\mathcal{D}^2 \setminus \mathcal{D}^2_* &= \{M_{(1,1)}, M_{(1,2)}, M_{(1,3)}, M_{(1,4)}\}
\end{aligned}$$

we have

$$\mathcal{D} = \{M_{(1,1)}, M_{(1,2)}, M_{(1,3)}, M_{(1,4)}\}.$$

Considering only the literals of the language of P, we obtain

$$\bigcap_{M \in \mathcal{D}} M = \{c, not\ u\}.$$

Hence $\{c, not\ u\}$ are the safe literals of $kerx_{MH_P}(P)$, all the other literals being suspicious. This result agrees with our intentions expressed in example 6.4.

Example 6.6. Let us compute the set of safe literals in the extended kernel of the following theory $P_{ic} = P \cup T_{ic}$, with respect to the MH_P semantics.

$$\begin{aligned}
\bot &\leftarrow not\ a \\
\bot &\leftarrow b \\
\neg a &\leftarrow not\ b \\
b &\leftarrow not\ \neg a \\
u &\leftarrow not\ \neg a, not\ b
\end{aligned}$$

Iteration 1

P_{ic} has two MH_P models.

$$\begin{aligned}
M_{(1)} &= \{not\ a, \neg a, not\ b, not\ \neg b, not\ u, not\ \neg u, \bot\} \\
H_{(1)} &= \{\neg a\} \\
S^{(1)} &= \{a\} \qquad S_{(1)} = S^{(1)} = \{a\} \\
M_{(2)} &= \{not\ a, not\ \neg a, b, not\ \neg b, not\ u, not\ \neg u, \bot\} \\
H_{(2)} &= \{b\} \\
S^{(2)} &= \{a, b\} \qquad S_{(2)} = S^{(2)} = \{a, b\} = \\
ker_{MH_P}(P) &= \{not\ a, not\ \neg b, not\ u, not\ \neg u\} \\
\mathcal{D}^1 &= \mathcal{D}^1_* = \{M_{(1)}, M_{(2)}\}.
\end{aligned}$$

Iteration 2

The programs $P_{(1)}, P_{(2)}$ corresponding to models $M_{(1)}, M_{(2)}$ are

$P_{(1)}$	$P_{(2)}$
$\bot \leftarrow not\ a$	$\bot \leftarrow not\ a$
$\bot \leftarrow b$	$\bot \leftarrow b$
$\neg a \leftarrow not\ b$	$\neg a \leftarrow not\ b$
$b \leftarrow not\ \neg a$	$u \leftarrow not\ \neg a, not\ b$
$u \leftarrow not\ \neg a, not\ b$	$b \leftarrow pos(b)$
$a \leftarrow pos(a)$	$pos(b) \leftarrow not\ neg(b)$
$pos(a) \leftarrow not\ neg(a)$	$neg(b) \leftarrow not\ pos(b)$
$neg(a) \leftarrow not\ pos(a)$	$a \leftarrow pos(a)$
	$pos(a) \leftarrow not\ neg(a)$
	$neg(a) \leftarrow not\ pos(a).$

We compute the MH_P models $M_{(i,j)}$ of each of the programs $P_{(i)}$, and the corresponding sets $S_{(i,j)}$ and $X_{(i,j)}$ (these last sets, which contain the switching atoms $pos(L), neg(L)$ pertaining to the model, are only indicated when $S_{(i,j)} \neq \emptyset$; they are boldfaced for an easier identification). We omit the computation of $P_{(1)}^{t-o}$ and $P_{(2)}^{t-o}$.

$$M_{(1,1)} = \{a, not\ \neg a, b, not\ \neg b, not\ u, not\ \neg u, not\ neg(a), pos(a), \bot\}$$
$$H_{(1,1)} = \{pos(a)\} \qquad \boldsymbol{X_{(1,1)} = \{pos(a)\}}$$
$$S^{(1,1)} = \{b\} \quad S_{(1,1)} = S^{(1,1)} \setminus S_{(1)} = \{b\}$$
$$M_{(1,2)} = \{not\ a, \neg a, not\ b, not\ \neg b, not\ u, not\ \neg u, neg(a), not\ pos(a), \bot\}$$
$$H_{(1,2)} = \{\neg a, neg(a)\}$$
$$S^{(1,2)} = \{a\} \quad S_{(1,2)} = S^{(1,2)} \setminus S_{(1)} = \emptyset$$
$$M_{(1,3)} = \{not\ a, not\ \neg a, b, not\ \neg b, not\ u, not\ \neg u, neg(a), not\ pos(a), \bot\}$$
$$H_{(1,3)} = \{b, neg(a)\} \qquad \boldsymbol{X_{(1,1)} = \{neg(a)\}}$$
$$S^{(1,3)} = \{a, b\} \quad S_{(1,3)} = S^{(1,3)} \setminus S_{(1)} = \{b\}$$

$$
\begin{aligned}
M_{(2,1)} &= \{a, \mathit{not}\ \neg a, b, \mathit{not}\ \neg b, \mathit{not}\ u, \mathit{not}\ \neg u, \mathit{not}\ \mathit{neg}(a), \mathit{not}\ \mathit{neg}(b), \mathit{pos}(a), \mathit{pos}(b), \bot\}\\
H_{(2,1)} &= \{\mathit{pos}(a), \mathit{pos}(b)\}\\
S^{(2,1)} &= \{b\} \quad S_{(2,1)} = S^{(2,1)} \setminus S_{(2)} = \emptyset\\
M_{(2,2)} &= \{\mathit{not}\ a, \mathit{not}\ \neg a, b, \mathit{not}\ \neg b, \mathit{not}\ u, \mathit{not}\ \neg u, \mathit{neg}(a), \mathit{not}\ \mathit{neg}(b), \mathit{not}\ \mathit{pos}(a), \mathit{pos}(b), \bot\}\\
H_{(2,2)} &= \{\mathit{neg}(a), \mathit{pos}(b)\}\\
S^{(2,2)} &= \{a, b\} \quad S_{(2,2)} = S^{(2,2)} \setminus S_{(2)} = \emptyset\\
M_{(2,3)} &= \{a, \mathit{not}\ a, \neg a, \mathit{not}\ \neg a, \mathit{not}\ b, \mathit{not}\ \neg b, u, \mathit{not}\ u, \mathit{not}\ \neg u, \mathit{not}\ \mathit{neg}(a), \mathit{neg}(b), \mathit{pos}(a),\\
&\qquad \mathit{not}\ \mathit{pos}(b), \bot\}\\
H_{(2,3)} &= \{\mathit{pos}(a), \mathit{neg}(b)\} \qquad \boldsymbol{X_{(2,3)} = \{\mathit{pos}(a), \mathit{neg}(b)\}}\\
S^{(2,3)} &= \{a, \neg a, u\} \quad S_{(2,3)} = S^{(2,3)} \setminus S_{(2)} = \{\neg a, u\}\\
M_{(2,4)} &= \{\mathit{not}\ a, \neg a, \mathit{not}\ b, \mathit{not}\ \neg b, \mathit{not}\ u, \mathit{not}\ \neg u, \mathit{neg}(a), \mathit{neg}(b), \mathit{not}\ \mathit{pos}(a), \mathit{not}\ \mathit{pos}(b), \bot\}\\
H_{(2,4)} &= \{\mathit{neg}(a), \mathit{neg}(b)\}\\
S^{(2,4)} &= \{a\} \quad S_{(2,4)} = S^{(2,4)} \setminus S_{(2)} = \emptyset
\end{aligned}
$$

Take the class $\mathcal{D}^2$ of models computed above is

$$\mathcal{D}^2 = \{M_{(1,1)}, M_{(1,2)}, M_{(1,3)}, M_{(2,1)}, M_{(2,2)}, M_{(2,3)}, M_{(2,4)}\},$$

while the class $\mathcal{D}^2_* \subseteq \mathcal{D}^2$ of models corresponding to nonempty $S_{(i,j)}$ is

$$\mathcal{D}^2_* = \{M_{(1,1)}, M_{(1,3)}, M_{(2,3)}\}.$$

As $\mathcal{D}^2_* \neq \emptyset$ a third iterative step takes place.

Iteration 3

Compute the programs $P_{(i,j)}$ corresponding to models $M_{(i,j)} \in \mathcal{D}^2_*$, by adding to $P_{(i)}$ the sets $X_{(i,j)}$, and expanding the literals in $S_{(i,j)}$.

$P_{(1,1)}$	$P_{(1,3)}$	$P_{(2,3)}$
$\bot \leftarrow \mathit{not}\ a$	$\bot \leftarrow \mathit{not}\ a$	$\bot \leftarrow \mathit{not}\ a$
$\bot \leftarrow b$	$\bot \leftarrow b$	$\bot \leftarrow b$
$\neg a \leftarrow \mathit{not}\ b$	$\neg a \leftarrow \mathit{not}\ b$	$a \leftarrow$
$u \leftarrow \mathit{not}\ \neg a, \mathit{not}\ b$	$u \leftarrow \mathit{not}\ \neg a, \mathit{not}\ b$	$\mathit{neg}(b) \leftarrow$
$a \leftarrow$	$\mathit{neg}(a) \leftarrow$	$\mathit{pos}(a) \leftarrow$
$\mathit{pos}(a) \leftarrow$	$b \leftarrow \mathit{pos}(b)$	$\neg a \leftarrow \mathit{pos}(\neg a)$
$b \leftarrow \mathit{pos}(b)$	$\mathit{pos}(b) \leftarrow \mathit{not}\ \mathit{neg}(b)$	$\mathit{pos}(\neg a) \leftarrow \mathit{not}\ \mathit{neg}(\neg a)$
$\mathit{pos}(b) \leftarrow \mathit{not}\ \mathit{neg}(b)$	$\mathit{neg}(b) \leftarrow \mathit{not}\ \mathit{pos}(b)$	$\mathit{neg}(\neg a) \leftarrow \mathit{not}\ \mathit{pos}(\neg a)$
$\mathit{neg}(b) \leftarrow \mathit{not}\ \mathit{pos}(b)$		$u \leftarrow \mathit{pos}(u)$
		$\mathit{pos}(u) \leftarrow \mathit{not}\ \mathit{neg}(u)$
		$\mathit{neg}(u) \leftarrow \mathit{not}\ \mathit{pos}(u)$

We compute the MH_P models $M_{(i,j,k)}$ of each of the programs $P_{(i,j)}$, and the corresponding sets $S^{(i,j,k)}$ and $X_{(i,j,k)}$.

$$M_{(1,1,1)} = \{a, not\ \neg a, b, not\ \neg b, not\ u, not\ \neg u, not\ neg(a), not\ neg(b), pos(a), pos(b), \bot\}$$
$$H_{(1,1,1)} = \{pos(b)\}$$
$$S^{(1,1,1)} = \{b\} \quad S_{(1,1,1)} = S^{(1,1,1)} \setminus (S_{(1,1)} \cup S_{(1)}) = \emptyset$$
$$M_{(1,1,2)} = \{a, not\ a, \neg a, not\ \neg a, not\ b, not\ \neg b, u, not\ u, not\ \neg u, not\ neg(a), neg(b), pos(a), not\ pos(b), \bot\}$$
$$H_{(1,1,2)} = \{neg(b)\} \qquad \boldsymbol{X_{(1,1,2)} = \{neg(b), pos(a)\}}$$
$$S^{(1,1,2)} = \{a, \neg a, u\} \quad S_{(1,1,2)} = S^{(1,1,2)} \setminus (S_{(1,1)} \cup S_{(1)}) = \{\neg a, u\}$$

Notice that in spite of being $\bot \leftarrow a, not\ a$ active with respect to $M_{(1,1,2)}$, the literal a is not in $S_{(1,1,2)}$, since it was already expanded.

$$M_{(1,3,1)} = \{not\ a, not\ \neg a, b, not\ \neg b, not\ u, not\ \neg u, not\ neg(b), neg(a), not\ pos(a), pos(b), \bot\}$$
$$H_{(1,3,1)} = \{pos(b)\}$$
$$S^{(1,3,1)} = \{a, b\} \quad S_{(1,3,1)} = S^{(1,3,1)} \setminus (S_{(1,3)} \cup S_{(1)}) = \emptyset$$
$$M_{(1,3,2)} = \{not\ a, \neg a, not\ b, not\ \neg b, not\ u, not\ \neg u, neg(a), neg(b), not\ pos(a), not\ pos(b), \bot\}$$
$$H_{(1,3,2)} = \{neg(b)\}$$
$$S^{(1,3,2)} = \{a\} \quad S_{(1,3,2)} = S^{(1,3,2)} \setminus (S_{(1,3)} \cup S_{(1)}) = \emptyset$$

$$M_{(2,3,1)} = \{a, not\ a, \neg a, not\ \neg a, not\ b, not\ \neg b, u, not\ \neg u, not\ neg(a), not\ neg(\neg a), neg(b), not\ neg(u), pos(a), pos(\neg a), not\ pos(b), pos(u), \bot\}$$
$$H_{(2,3,1)} = \{pos(\neg a), pos(u)\}$$
$$S^{(2,3,1)} = \{a, \neg a\} \quad S_{(2,3,2)} = S^{(2,3,2)} \setminus (S_{(2,3)} \cup S_{(2)}) = \emptyset$$
$$M_{(2,3,2)} = \{a, not\ \neg a, not\ b, not\ \neg b, u, not\ \neg u, not\ neg(a), neg(\neg a), neg(b), not\ neg(u), pos(a), not\ pos(\neg a), not\ pos(b), pos(u)\}$$
$$H_{(2,3,2)} = \{neg(\neg a), pos(u)\}$$
$$S^{(2,3,2)} = \emptyset = \quad S_{(2,3,2)} = \emptyset$$
$$M_{(2,3,3)} = \{a, not\ a, \neg a, not\ \neg a, not\ b, not\ \neg b, not\ u, not\ \neg u, not\ neg(a), not\ neg(\neg a), neg(b), neg(u), pos(a), pos(\neg a), not\ pos(b), not\ pos(u), \bot\}$$
$$H_{(2,3,3)} = \{pos(\neg a), neg(u)\}$$
$$S^{(2,3,3)} = \{a, \neg a\} \quad S_{(2,3,3)} = S^{(2,3,3)} \setminus (S_{(2,3)} \cup S_{(2)}) = \emptyset$$
$$M_{(2,3,4)} = \{a, not\ \neg a, not\ b, not\ \neg b, not\ u, not\ u, not\ \neg u, neg(a), not\ neg(\neg a), neg(b), neg(u), pos(a), not\ pos(\neg a), not\ pos(b), not\ pos(u), \bot\}$$
$$H_{(2,3,4)} = \{neg(\neg a), neg(u)\}$$
$$S^{(2,3,4)} = \emptyset \quad S_{(2,3,4)} = \emptyset$$

The class $\mathcal{D}^3$ of models computed in iteration step 3 is,

$$\mathcal{D}^3 = \{M_{(1,1,1)}, M_{(1,1,2)}, M_{(1,3,1)}, M_{(1,3,2)}, M_{(2,3,1)}, M_{(2,32)}, M_{(2,3,3)}, M_{(2,3,4)}\},$$

while the class $\mathcal{D}^3_* \subseteq \mathcal{D}^3$ is

$$\mathcal{D}^3_* = \{M_{(1,1,2)}\}.$$

As $\mathcal{D}^3_* \neq \emptyset$ a fourth iterative step takes place.

Iteration 4

Programs to consider for iteration 4 are $P_{(1,1,2)}$. As $S_{(1,1,2)} = \{\neg a, u\}$ and $X_{(1,1,2)} = \{neg(b), pos(a)\}$, $P_{(1,1,2)}$ is the same as $P_{(2,3)}$. Notice that $S_{(1)} \cup S_{(1,1)} \cup S_{(1,1,2)} = \{a, \neg a, b, u\}$ and hence, not only no new models appear in the iteration step 4, as also $\mathcal{D}^4_* = \emptyset$. Thus the iterative part of the algorithm is finished. The process continues at the *last step*.

Last step

We have

$$\begin{aligned}
&\mathcal{D}^1 \setminus \mathcal{D}^1_* = \emptyset \\
&\mathcal{D}^2 \setminus \mathcal{D}^2_* = \{M_{(1,2)}, M_{(2,1)}, M_{(2,2)}, M_{(2,4)}\} \\
&\mathcal{D}^3 \setminus \mathcal{D}^3_* = \{M_{(1,1,1)}, M_{(1,3,1)}, M_{(1,3,2)}, M_{(2,3,1)}, M_{(2,3,2)}, M_{(2,3,3)}, M_{(2,3,4)}\},
\end{aligned}$$

$\mathcal{D}^4 \setminus \mathcal{D}^4_*$ being not important, and thus

$$\begin{aligned}
\mathcal{D} = \{&M_{(1,2)}, M_{(2,1)}, M_{(2,2)}, M_{(2,4)}, M_{(1,1,1)}, M_{(1,3,1)}, M_{(1,3,2)}, M_{(2,3,1)}, M_{(2,3,2)}, M_{(2,3,3)}, \\
&M_{(2,3,4)}\}
\end{aligned}$$

Considering only the literals of the language of P, we obtain

$$\bigcap_{M \in \mathcal{D}} M = \{not\ \neg b, not\ \neg u\},$$

and hence those are the safe literals of $kerx_{MH_P}(P_{ic})$. All the other literals are suspicious.

6.3 Query Answering with Revision and Inconsistency Propagation Detection

Let $P_{ic} = P \cup T_{ic}$ be a finite ground inconsistent theory that represents a database. With the revision and inconsistency propagation detection tools presented in this chapter, several ways to answer queries to the database are now at our disposal, besides the ones already presented in section 5.5. In this section, we use the simple relevance property of MH_P for query answering in a brave reasoning fashion.

Let then Q be a conjunction of objective and/or default literals that represents a query to the database P_{ic}. Consider the subset P_Q of P_{ic} computed as follows.

1. Let $P_{Q,0} = Rel_P(Q)$ where $Rel_P(Q)$ is the union of the subprograms of P that are relevant to each objective literal involved in Q.
2. Join to $P_{Q,0}$ all the integrity constraints of T_{ic} that depend on objective literals of $P_{Q,0}$ language. Let $P_{Q,0,\perp}$ be the resulting program.
3. Join to $P_{Q,0,\perp}$ all the subprograms of P that are relevant to each objective literal involved in $P_{Q,0,\perp}$. Let $P_{Q,1}$ be the resulting program.
4. Join to $P_{Q,1}$ all the integrity constraints of T_{ic} that depend on objective literals of $P_{Q,1}$ language. Let $P_{Q,1,\perp}$ be the resulting program.
5. Repeat this procedure until an order $n \in \mathbb{N}$ is reached, such that $P_{Q,n-1,\perp} = P_{Q,n,\perp}$. Let $P_Q = P_{Q,n,\perp}$ be the resulting program.

Notice that this procedure terminates, since T_{ic} is finite and the sequence $P_{Q,0,\perp}, P_{Q,1,\perp}, \cdots, P_{Q,k,\perp}, \cdots$ is monotonically increasing with respect to set inclusion.

The theory P_Q may be used to answer the query Q in a number of ways, among which are the following.

Brave Reasoning Fashion

1. (Considering only revision) If Q is satisfied by at least one model of a revised program of P_Q, then the answer should be "yes". If Q is not satisfied by at least one model of some revised program, then the answer should be "no".
2. (Considering only suspicion detection) If Q is satisfied by at least one model of P_Q and no objective literal involved in Q is suspicious with respect to that model, then the answer should be "yes". If some literals involved in Q are suspicious, then the answer should be "yes". If Q is satisfied by no model of P_Q, then the answer should be "no".

All the answers above, may come together with additional comments that make clear the *Six* valuation of the conjunction that represents the query.

Other strategies for query answering may be envisaged by mixing revision and suspicion detection, e.g., adding suspicion information to the models of revised programs. The types of query answering presented in section 5.5, may also be improved with the procedures for revision and inconsistency propagation detection provided in this chapter.

6.4 Final Remarks

The revising and inconsistency propagation detection procedures, may be envisaged has having, in a certain way, overlapping goals. That is, given an inconsistent constrained theory $P_{ic} = P \cup T_{ic}$, we could take as safe literals in $kerx_{MH_P}(P)$ all of these literals that also belong to the intersection of the extended kernels of all obtained revised theories. We could also try to get revised theories from the expansions whose semantics contain the smallest sets of active integrity constraints. Notwithstanding it is worth to point out that the inconsistency propagation detection process we present in this chapter has a "more skeptical" character than the revision process, would the latter be used for inconsistency propagation detection as referred above. The following example clarifies this issue.

Example 6.7. Consider the following constrained theory $P_{ic} = P \cup T_{ic}$.

$$\begin{aligned} \bot &\leftarrow a, b \\ c &\leftarrow a \\ a &\leftarrow b \\ b &\leftarrow \end{aligned}$$

P_{ic} has a single MH_P model, $\{a, b, c, not\ \neg a, not\ \neg b, not\ \neg c, \bot\}$, and hence $kerx_{MH_P} = \{a, b, c, not\ \neg a, not\ \neg b, not\ \neg c\}$. Using the inconsistency propagation detection algorithm, it is immediate to conclude that the subset of suspicious literals in the kernel is $kerx_{MH_P} = \{a, b, c\}$, meaning that c is a suspicious literal. Meanwhile, all the revised theories have *not* c in their kernels, since the revision process produces models that do not activate the constraint $\bot \leftarrow a, b$, by deleting one of the rules $\{a \leftarrow b,\ b \leftarrow\}$, which falsifies c. That is, c is not suspicious if we consider the information in the revised theories' kernels.

Chapter 7

Conclusions and Future Work Proposals

We sum up the results obtained in this work, together with some criticism, and point out paths to future work.

The main goal of this work was to implement a semantics for extended normal logic programs, that could be existential whilst amenable for inconsistency revision through abduction of revisables. The MH_P semantics defined in chapter 5 fulfills this intentions. It is an existential total models paraconsistent semantics that treats loops as choice devices, which permits to solve problems that are, apparently, not in the reach of answer sets semantics. In this respect, it is an open issue whether there is a transformation from normal logic programs into normal logic programs, such that the MH models of the initial program can be obtained through the answer sets of the transformed program. The author of this work ventures that it is not possible to obtain such a transformation. Meanwhile the MH_P models computation, via the balanced layered remainder, motivates the search for a reduction system for extended normal logic programs, that could permit the MH_P semantics computation using the resulting remainder, eliminating thus the need for the $t - o$ transformation. Such a system would have a theoretical interest, despite of being not clear whether any computational advantage could stem from this "yet to be found" alternative process.

As companions of the paraconsistent MH_P semantics, a process for program revision and a process for inconsistency propagation detection are presented in chapter 6. The program revision is fulfilled by resorting to declarative debugging, eliminating rules of the program or modifying the CWA valuations of certain literals without a rule. This approach may be envisaged as a revision by reductio ad absurdum. The revision procedure accepts as much as possible the initial theory, meaning that only those rules whose heads are involved in the bodies of active constraints are considered for deletion. This bears a relation with the philosophic *principle of charity*, if we envisage a constrained theory as expressing the convictions of an agent. It is worth to say that other strategies for rule deletion may be considered. The programmer may refine the tags used as arguments of the predicative functions $del()$, in order to delete specific rules, instead of deleting all the rules with a certain head. He/she can also choose to throw into the program revisables that lead to the deletion of other rules, not only the ones whose heads appear in active constraints. The programmer may define sets of revisables, ordered in accordance to certain preferential criteria, and choose the revised programs in accordance. And so on. Revision processes tend to be highly dependent on the idiosyncrasies of the programmer, or on the particularities of the contexts in which they are used. Our revision proposal intentions are not, for sure, to present a revision process that works well in all contexts, but rather to exemplify how the MH_P is open to revision purposes. The inconsistency propagation detection process outlined in this work, considers detection of inconsistencies "upwards" in the program layering, meaning that only literals defined in the same layer, or in layers above those of literals appearing in the bodies of active constraints, are considered for expansion. Something analogous also happens with the revision process, where the rules considered for deletion are in the same layers of rules that define the literals involved in the bodies of active constraints. This "not downwards" character makes both processes in a certain way incomplete. As an example of this limitation, let us face revision and inconsistency propagation detection of the theory $P_{ic} = P \cup T_{ic}$

$$\begin{aligned} \bot &\leftarrow a \\ a &\leftarrow b \\ b &\leftarrow \end{aligned}$$

whose single MH_P model is $M = \{a, b, \bot\}$. The only resulting revised theory is

$$\begin{aligned} \bot &\leftarrow a \\ b &\leftarrow \end{aligned}$$

and the only safe literal in $kerx_{MH_P}(P)$ is b. Now, a "downwards" analysis could consider rule $b \leftarrow$ as a candidate for deletion in a revision process, and atom b as a candidate for expansion in an inconsistency propagation detection process. While these options seem arguably reasonable, we do not consider them in our work. A systematization of this "downwards" type of approach may be far from trivial, due to the necessity of handling loops. Even if in the program above it seems an easy job to include the "downwards" option into our procedures, that is because the program is a very simple stratified one. Finding a reasonable way, in the computational point of view, to make our procedures act "downwards", is a future work proposal. Another path for future work might be the design of a paraconsistent semantics, MH_P^*, that extends to broader constraint theories the MH_P capability to detect support on inconsistency. Also a valuable path of investigation, in our opinion, is the use of MH_P and of the procedures proposed in chapter 6 to implement systems for dealing with updating and evolving programs, by adapting those tools to the approaches outlined in [ALP$^+$98] and [ABLP02], thus extending the work there presented. The revision of sequences obtained as a result of updates, may allow choices or preferences on revisions (e.g., the revisions could address the entire sequence, the initial state or the final state). Other combinations could also be regarded. Considering several revision solutions, eventually complemented with inconsistency propagation detection, may also be a means of prospecting future scenarios.

Some suggestions for future work already mentioned, are summarized below:

1. Find whether there is a transformation from normal logic programs to normal logic programs, such that the MH models of the initial program can be obtained through the answer sets of the transformed program.[1]

2. Find whether there is a reduction system for extended normal logic programs, that allows the computation of MH_P models using a remainder obtained through the performance of that system, eliminating thus the need for the $t-o$ transformation.

3. Endow the revision and inconsistency propagation detection algorithms with "downwards" performing capabilities, in some way to be defined.

4. Endow the MH_P semantics with inconsistency propagation detection with respect to broad constraint theories, thus eliminating the need for the safeness algorithm presented in chapter 6.

5. Use the MH_P semantics and the procedures developed in chapter 6 to handle updating tasks and to deal with evolving logic theories.

In chapter 4 a number of results concerning the characterization of 2-valued conservative extensions of the stable model semantics were presented. These results allow a precise characterization of the MH and the MH_P semantics on the properties of relevance and cumulativity. Some relations among strong and weak properties, concerning semantics from the ASM^h and ASM^m families, were settled. We have shown that the definition of these families reveals an universe of semantics, for which the cumulativity definition expresses a relation among sets of models instead of a relation among sets of atoms. As a consequence of this result, the study of cumulativity inside ASM^h and ASM^m becomes an easier job. The structural properties of defectivity and excessiveness may turn this work even easier: the detection of cumulativity failure is faced not as a relation among sets of models, but as the relation between models and the structure (layering) of logic programs, defined by the decomposition of models over the programs structures. This approach allows to relate the failure of existence to the failure of cautious monotony for semantics of the above families, i.e., if a semantics is not existential, then it cannot be cautious monotonic. In the case of the SM semantics, which pertains to both families, this result is, to the best of our knowledge, new. This type of results show the potential of a structural approach, so to say, to the study of semantics formal properties. It seems thus reasonable to try to extend this approach to embrace other than 2-valued models semantics, and other than relevance and cumulativity weak and strong properties. Some suggestions for future work on this subject follow.

[1]This open problem is already mentioned in [Pin11]. A solution for it would also have implications in the MH_P semantics computation.

6. Find sets of structural properties whose verification is equivalent to the verification of cautious monotony or cut. The existence of such sets would turn into an easier job, both obtaining cumulative semantics and to detect the failure of cumulativity. This might demand a sharper definition of layering.
7. Find a structural characterization for other than relevance and cumulativity weak and strong properties.
8. Extend the type of approach undertaken in chapter 4 to other than 2-valued models semantics.
9. Extend the type of approach undertaken in chapter 4 to non-noetherian logic programs; this might need a looser definition of layering, by dropping the well-ordering demand on the set of labels (see definition 2.11).
10. Complete table 4.1, either by finding adequate semantics, or by show they do not exist.
11. Find a ASM^h cumulative semantics.
12. Find whether there is a ASM^h or ASM^m cumulative semantics, that is not defined by means of an iterative procedure like the one used to define $Cyan$ (see appendix B.3), that is, a semantics where each model is computed a deterministic number of times.

The 12 proposals for future work here presented, may eventually be interesting enough, some of them even defying, for anyone interested in these subjects.

Bibliography

[AB94] Krzysztof R. Apt and Roland N. Bol. Logic programming and negation: A survey. *J. Log. Program.*, pages 9–71, 1994.

[ABLP02] José Júlio Alferes, Antonio Brogi, João Alexandre Leite, and Luís Moniz Pereira. Evolving logic programs. In *JELIA'02*, pages 50–61, 2002.

[ABW88] K. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of deductive databases and logic programming*, pages 89–142, Los Altos, CA, 1988. Morgan Kaufmann.

[ADP95] José Júlio Alferes, Carlos Viegas Damásio, and Luís Moniz Pereira. A logic programming system for nonmonotonic reasoning. *J. Autom. Reasoning*, 14(1):93–147, 1995.

[Alf93] José Júlio Alferes. *Semantics of Logic Programs with Explicit Negation.* PhD thesis, Universidade Noval de Lisboa, October 1993.

[ALP$^+$98] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic logic programming. In *APPIA-GULP-PRODE'98*, pages 393–408, 1998.

[AP96] José Júlio Alferes and Luís Moniz Pereira. *Reasoning with Logic Programming*, volume 1111 of *Lecture Notes in Computer Science*. Springer, 1996.

[AP13] Mário Abrantes and Luís Moniz Pereira. An abductive paraconsistent semantics - MHp. In *INAP'13*, 2013.

[Ari02] Ofer Arieli. Paraconsistent declarative semantics for extended logic programs. *Ann. Math. Artif. Intell*, 36(4):381–417, 2002.

[BDFZ01] Stefan Brass, Jürgen Dix, Burkhard Freitag, and Ulrich Zukowski. Transformation-based bottom-up computation of the well-founded model. *TPLP*, pages 497–538, 2001.

[BG94] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *J. Log. Program.*, pages 73–148, 1994.

[BS89] Howard A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theor. Comput. Sci.*, pages 135–154, 1989.

[CKRP73] A. Colmerauer, H. Kanoui, P. Roussel, and R. Passero. Un système de communication homme-machine en Francais. Technical report, Groupe de Recherche en Intelligence Artificielle, Universitè d'Aix-Marseille II, 1973.

[Cos95] Stefania Costantini. Contributions to the stable model semantics of logic programs with negation. *Theoretical Computer Science*, 149(2):231–255, 2 October 1995.

[Dam96] Carlos Viegas Damásio. A survey of paraconsistent semantics for logic programs. 1996.

[dCBB95] Newton C. A. da Costa, Jean-Yves Béziau, and Otávio A. S. Bueno. Aspects of paraconsistent logic. *Logic Journal of the IGPL*, pages 597–614, 1995.

[Dix95a] Jürgen Dix. A classification theory of semantics of normal logic programs: I. strong properties. *Fundam. Inform*, 22(3):227–255, 1995.

[Dix95b] Jürgen Dix. A classification theory of semantics of normal logic programs: II. weak properties. *Fundam. Inform*, 22(3):257–288, 1995.

[Dix96] Jürgen Dix. Semantics of logic programs: Their intuitions and formal properties. an overview. In *Logic, Action, and Information*, pages 241–327, 1996.

[DK02] Marc Denecker and Antonis C. Kakas. Abduction in logic programming. In *Computational Logic: Logic Programming and Beyond'02*, pages 402–436, 2002.

[DP97] Carlos Viegas Damásio and Luís Moniz Pereira. A paraconsistent semantics with contradiction support detection. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *LPNMR*, volume 1265 of *Lecture Notes in Computer Science*, pages 224–243. Springer, 1997.

[DP98] Carlos Viegas Damásio and Luís Moniz Pereira. A survey of paraconsistent semantics for logic programs. *Handbook of defeasible reasoning and uncertainty management systems: volume 2: reasoning with actual and potential contradictions*, pages 241–320, 1998.

[DSTW12] J. Delgrande, T. Schaub, H. Tompits, and S. Woltran. A model-theoretic approach to belief change in answer set programming. *ACM Transactions on Computational Logic*, 2012. To appear.

[Gel93] A. Van Gelder. The alternating fixpoint of logic programs with negation. *J. of Comp. System Sciences*, 47(1):185–221, 1993.

[Gin88] Matthew L. Ginsberg. Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, pages 265–316, 1988.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.

[GL90] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In *ICLP*, pages 579–597, 1990.

[GRS91] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *J. of ACM*, 38(3):620–650, 1991.

[GSC98] Alejandro García, Guillermo R. Simari, and Carlos I. Chesñevar. An argumentative framework for reasoning with inconsistent and incomplete information. In *Proceedings of the ECAI'98 Workshop on Practical Reasoning and Rationality*, pages 13–20. Brighton, UK, August 1998.

[KS90] Robert A. Kowalski and Fariba Sadri. Logic programs with exceptions. In *ICLP'90*, pages 598–613, 1990.

[Lif08] Vladimir Lifschitz. What is answer set programming? In *AAAI'08*, pages 1594–1597, 2008.

[Llo87] John Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[LMR92] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, 1992.

[Mcc59] John Mccarthy. Programs with common sense. In *Semantic Information Processing*, pages 403–418. MIT Press, 1959.

[PA92] Luís Moniz Pereira and José Júlio Alferes. Well founded semantics for logic programs with explicit negation. In *ECAI'92*, pages 102–106, 1992.

[PAA92] Luís Moniz Pereira, José Júlio Alferes, and Joaquim Nunes Aparício. Adding closed world assumptions to well founded semantics. In *FGCS*, pages 562–569, 1992.

[PDA93a] Luís Moniz Pereira, Carlos Viegas Damásio, and José Júlio Alferes. Debugging by diagnosing assumptions. In Peter Fritzon, editor, *Automated and Algorithmic Debugging, First International Workshop, AADEBUG'93*, volume 749 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 3–5 May 1993.

[PDA93b] Luís Moniz Pereira, Carlos Viegas Damásio, and José Júlio Alferes. Diagnosis and debugging as contradiction removal. In *LPNMR*, pages 316–330, 1993.

[Pin11] Alexandre Miguel Pinto. *Every normal logic program has a 2-valued semantics: theory, extensions, applications, implementations.* PhD thesis, Universidade Nova de Lisboa, July 2011. Published in the December issue of the ALP newsletter:http://www.cs.nmsu.edu/ALP/2012/12/doctoral-dissertation-every-normal-logic-program-has-a-2-valued-semantics-theory-extensions-applications-implementations/.

[PP09] Luís Moniz Pereira and Alexandre Miguel Pinto. Layer supported models of logic programs. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *LPNMR*, volume 5753 of *Lecture Notes in Computer Science*, pages 450–456. Springer, 2009.

[PP11] Alexandre Miguel Pinto and Luís Moniz Pereira. Each normal logic program has a 2-valued minimal hypotheses semantics. *INAP 2011, CoRR*, abs/1108.5766, 2011.

[Prz90a] Teodor Przymusinski. Well-founded Semantics Coincides With Three-Valued Stable Semantics. *Fundamenta Informaticae*, XIII:445–463, 1990.

[Prz90b] Teodor C. Przymusinski. Extended stable semantics for normal and disjunctive programs. In *ICLP'90*, pages 459–477, 1990.

[PW89] David Pearce and Gerd Wagner. Logic programming with strong negation. In Peter Schroeder-Heister, editor, *ELP*, volume 475 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 1989.

[Rei78] Raymond Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum, New York / London, 1978.

[Sak92] Chiaki Sakama. Extended well-founded semantics for paraconsistent logic programs. In *FGCS*, pages 592–599, 1992.

[Swi99] Terrance Swift. Tabling for non-monotonic programming. *Ann. Math. Artif. Intell*, 25(3-4):201–240, 1999.

[SZ91] Domenico Saccà and Carlo Zaniolo. Partial models and three-valued models in logic programs with negation. In *LPNMR'91*, pages 87–101, 1991.

[Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, pages 146–160, 1972.

[vEK76] Maarten H. van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976.

[Wag91] Gerd Wagner. A database needs two kinds of negation. In *MFDBS*, pages 357–371, 1991.

Appendix A

Refers to Chapter 3

Definitions in subsection 3.2.3

Definition A.1. Positive reduction (adapted from [BDFZ01]) Let P_1 and P_2 be two ground normal logic programs. Program P_2 results from P_1 by *positive reduction*, $P_1 \mapsto_P P_2$ iff there is a rule $r \in P_1$ and a default literal $not\ b \in Body(r)$ such that $b \notin Heads(P_1)$, and $P_2 = (P_1 \setminus \{r\}) \cup \{Head(r) \leftarrow (Body(r) \setminus \{not\ b\})\}$.

Definition A.2. Negative reduction. (adapted from [BDFZ01]) Let P_1 and P_2 be two ground normal logic programs. Program P_2 results from P_1 by *negative reduction*, $P_1 \mapsto_N P_2$ iff there is a rule $r \in P_1$ and a default literal $not\ b \in Body(r)$ such that $b \in Facts(P_1)$, and $P_2 = P_1 \setminus \{r\}$.

Definition A.3. Success. (adapted from [BDFZ01]) Let P_1 and P_2 be two ground normal logic programs. Program P_2 results from P_1 by *success*, $P_1 \mapsto_S P_2$, iff there is a rule $r \in P_1$ and a fact $b \in Facts(P_1)$ such that $b \in Body(r)$, and $P_2 = (P_1 \setminus \{r\}) \cup \{Head(r) \leftarrow (Body(r) \setminus \{b\})\}$.

Definition A.4. Failure. (adapted from [BDFZ01]) Let P_1 and P_2 be two ground normal logic programs. Program P_2 results from P_1 by *failure*, $P_1 \mapsto_F P_2$, iff there is a rule $r \in P_1$ and a positive literal $b \in Body(r)$ such that $b \notin Heads(P_1)$, and $P_2 = P_1 \setminus \{r\}$.

Definition A.5. Loop Detection. (adapted from [BDFZ01]) Let P_1 and P_2 be two ground normal logic programs. Program P_2 results from P_1 by *loop detection*, $P_1 \mapsto_L P_2$, iff there is a set $\mathcal{A}$ of ground atoms such that:

1. For each rule $r \in P_1$, if $Head(r) \in \mathcal{A}$, then $Body(r) \cap \mathcal{A} \neq \emptyset$;
2. $P_2 := \{r \in P_1 | Body(r) \cap \mathcal{A} = \emptyset\}$.

Appendix B

Refers to Chapter 4

B.1 Refers to Section 4.3

Proofs for the $\mathrm{ASM^h}$ and $\mathrm{ASM^m}$ Families of Semantics

Proof. (of proposition 4.4)
"$\Rightarrow$"
If SEM is not cautious monotonic, then there is a program P, a set $S \subseteq ker_{SEM}(P)$ and an atom $u \in ker_{SEM}(P)$ such that $u \notin ker_{SEM}(P \cup S)$, by definition 4.12. This immediately implies the existence of a model $M \in SEM(P \cup S)$ such that $u \notin M$, and hence $M \notin SEM(P)$.
"$\Leftarrow$"
We first prove this condition for semantics SEM of the ASM^h family.
Let P be a finite ground normal logic program and $S \subseteq ker_{SEM}(P)$. Consider an interpretation $M \in SEM(P \cup S)$, $M \notin SEM(P)$. Let $M = \{a_1, a_2, \cdots, a_m, not\, b_1, not\, b_2, \cdots, not\, b_n\}$, where the Herbrand base of P is $\{a_1, a_2, \cdots, a_m, b_1, b_2, \cdots, b_n\}$. We use program P to build a new program P' whose language contains the atom u, that does not belong to the language of P, and show that $u \in ker_{SEM}(P')$ and $u \notin ker_{SEM}(P' \cup S)$, where $S \subseteq ker_{SEM}(P')$, thus revealing that SEM fails cautious monotony. Let P' be the program

$$\begin{aligned}
&P\\
&c_j \leftarrow b_j\\
&d_j \leftarrow not\ c_j\\
&v \leftarrow a_1, \cdots, a_m, d_1, \cdots, d_n\\
&u \leftarrow not\ v
\end{aligned}$$

where P (let $P = P'^{\leq T}$) is an abbreviation for the set of rules of the program P; $(c_j \leftarrow b_j)$ and $(d_j \leftarrow not\ c_j)$ are schemes of rules, representing individual rules for each $1 \leq j \leq n$; rule $(v \leftarrow a_1, \cdots, a_m, d_1, \cdots, d_n)$ has the body formed by the conjunction of all a_i and d_j, $1 \leq i \leq m$, $1 \leq j \leq n$; the atoms v, u, c_j and d_j do not belong to the language of P. Now the set of all atoms involved in affixes of the models of P' is defined by the rules of $P'^{\leq T}$, due to the definition of ASM^h – notice that no c_j may belong to the affix of a SEM model of P', since such an affix would not be minimal. Thus the T-segment decomposition of any SEM model of P', $M_{\leq T} = M^+_{\leq T} \cup not\ M^+_{\leq T}$ is such that $(M^+_{\leq T}, M^-_{\leq T})$

is a SEM model of P, which means that $u \in ker_{SEM}(P')$, because all models of P' falsify $Body(v \leftarrow a_1, \cdots a_m, d_1, \cdots d_n)$. Let now $S \subseteq ker_{SEM}(P)$. Then $S \subseteq ker_{SEM}(P')$ and $SEM(P' \cup S)$ contains the model

$$M' = M \cup \{not\ c_1, \cdots, not\ c_n, d_1, \cdots, d_n, v, not\ u\}$$

and thus $u \notin ker_{SEM}(P' \cup S)$, meaning that SEM is not cautious monotonic. To see that this result is also valid for the ASM^m family, we just need to repeat the above reasoning considering instead program P'

$$\begin{array}{c} P \\ u \leftarrow M_k^+ \end{array}$$

where $u \leftarrow M_k^+$ represents a scheme of rules, one per model $M_k \in SEM(P)$ – let P have t SEM models, $1 \leq k \leq t$, M_k^+ being the conjunction of all positive literals in M_k. Clearly $u \in ker_{SEM}(P')$ and $u \notin ker_{SEM}(P' \cup S)$, since the model $M' \in SEM(P' \cup S)$, $M' = M \cup \{not\ u\}$ falsifies the body of any rule generated by the scheme $u \leftarrow M_k^+$.[1] $\square$

Proof. (of proposition 4.6)
"$\Rightarrow$"
If SEM is not cut, then there is a program P, a set $S \subseteq ker_{SEM}(P)$ and an atom $u \in ker_{SEM}(P \cup S)$, such that $u \notin ker_{SEM}(P)$, by definition 4.13. This immediatly implies the existence of a model $M \in SEM(P)$ such that $u \notin M$, and hence $M \notin SEM(P \cup S)$.
"$\Leftarrow$"
We first prove this condition for semantics SEM of the ASM^h family.
Let P be a normal logic program, $S \subseteq ker_{SEM}(P)$. Let $SEM(P \cup S) = \{M^1, M^2, \cdots, M^k\}$, where $k \in \mathbb{N}$ and

$$M^i = \{a_1^i, a_2^i, \cdots, a_{m_i}^i, not\ b_1^i, not\ b_2^i, \cdots, not\ b_{n_i}^i\},$$

the Herbrand base of P being, say $\{a_1^1, a_2^1, \cdots, a_{m_1}^1, b_1^1, b_2^1, \cdots, b_{n_1}^1\}$. Let also $M \in SEM(P)$, $M \notin SEM(P \cup S)$. We use program P to build a new program P' whose language contains the atom u, that does not belong to the language of P, and show that $u \notin ker_{SEM}(P')$ and $u \in ker_{SEM}(P' \cup S)$ where $S \subseteq ker_{SEM}(P')$, thus revealing that SEM is not cut. Let P' be the program

$$\begin{array}{l} P \\ c_j^i \leftarrow b_j^i \\ d_j^i \leftarrow not\ c_j^i \\ u \leftarrow a_1^i, \cdots, a_{m_i}^i, d_1^i, \cdots, d_{n_i}^i \end{array}$$

where P (let $P = P'^{\leq T}$) is an abbreviation for the set of rules of the program P; $(c_j^i \leftarrow b_j^i)$ and $(d_j^i \leftarrow not\ c_j^i)$ are schemes of rules representing individual rules for each pair of natural numbers (i, j), $1 \leq i \leq k$ and $1 \leq j \leq n_i$; $(u \leftarrow a_1^i, \cdots, a_{m_i}^i, d_1^i, \cdots, d_{n_i}^i)$ is a scheme of rules representing a rule for each $1 \leq i \leq k$; the atoms c_j^i, d_j^i, u, do not belong to the

[1]Notice that if $S \subseteq ker_{SEM}(P)$, then no minimal model N of $P \cup S$ may be such that $M_k^+ \subsetneq N^+$, for some k, otherwise N would not be a minimal model of $P \cup S$.

language of P. Now the set of all atoms involved in affixes of models of P' is defined by the rules of $P'^{\leq T}$, due to the definition of ASM^h – notice that no c_j may pertain to the affix of a SEM model of P', since such an affix would not be minimal. Thus the T-segment decomposition of any SEM model of P', $M_{\leq T} = M^+_{\leq T} \cup not\ M^+_{\leq T}$, is such that $(M^+_{\leq T}, M^-_{\leq T})$ is a SEM model of P, which means that $u \notin ker_{SEM}(P')$ since the unique model of program $M^+ \cup \{c^i_j \leftarrow b^i_j,\ d^i_j \leftarrow not\ c^i_j,\ u \leftarrow a^i_1, \cdots, a^i_{m_i}, d^i_1, \cdots, d^i_{n_i}\}$ does not satisfy $Body(u \leftarrow a^i_1, \cdots, a^i_{m_i}, d^i_1, \cdots, d^i_{n_i})$. As $ker_{SEM}(P) \subseteq ker_{SEM}(P')$, we have $S \subseteq ker_{SEM}(P')$. Each model in $SEM(P' \cup S)$ contains the atom u since, by design of P', $M^{i+} \cup \{c^i_j \leftarrow b^i_j,\ d^i_j \leftarrow not\ c^i_j,\ u \leftarrow a^i_1, \cdots, a^i_{m_i}, d^i_1, \cdots, d^i_{n_i}\}$ has a single model containing u, for any $1 \leq i \leq k$. Thus $u \in ker_{SEM}(P' \cup S)$, meaning that SEM is not cut. To see that this result is also valid for semantics of the ASM^m family, we just need to repeat the above reasoning considering instead the program P'

$$\begin{aligned} &P \\ &u \leftarrow M^{i+} \end{aligned}$$

where $u \leftarrow M^{i+}$ represents a scheme of rules, one per model $M^i \in SEM(P \cup S)$, $1 \leq i \leq k$, M^{i+} being the conjunction of all positive literals in M^i. Clearly $u \in ker_{SEM}(P' \cup S)$. Also $u \notin ker_{SEM}(P')$, since model $M' = M \cup \{not\ u\}$, $M \in SEM(P)$, $M \notin SEM(P \cup S)$ falsifies the body of any rule generated by the scheme $u \leftarrow M^{i+}$.[2] □

Proof. (of proposition 4.8)
Let P a normal logic program such that $SEM(P) \neq \emptyset$, $S \subseteq ker_{SEM}(P)$ and $SEM(P \cup S) = \emptyset$. We use program P to build a new program Q whose language contains the atom u, that does not belong to the language of P, and show that $u \notin ker_{SEM}(Q)$ and $u \in ker_{SEM}(Q \cup S)$ where $S \subseteq ker_{SEM}(Q)$, thus revealing that SEM is not cut. Let Q be the program

$$\begin{aligned} a &\leftarrow not\ b \\ b &\leftarrow not\ a \\ &P \\ Heads(P) &\leftarrow a \\ u &\leftarrow a, Heads(P) \end{aligned}$$

where P is an abbreviation for the set of rules of the program P; the scheme of rules $(Heads(P) \leftarrow a)$ represents a rule r per atom of $Heads(P)$, $Head(r) \in Heads(P)$; $(u \leftarrow a, Heads(P))$ is a rule whose body contains the conjunction of a and all atoms in $Heads(P)$; the atoms a, b, u do not belong to the language of P. Now Q has the SEM models $M^+ = \{a, u\} \cup Heads(P)$ and $N^+_Q = N^+_* \cup \{b\}$, for each SEM model N_* of P. Clearly $u \notin ker_{SEM}(Q)$ and $S \subseteq ker_{SEM}(Q)$. The program $Q \cup S$ has the single model $M^+ = \{a, u\} \cup Heads(P)$, since model $\{not\ a, b\} \in SEM(Q^{\leq 1})$ supports no SEM model of $Q \cup S$ – otherwise it would be $SEM(P \cup S) \neq \emptyset$. As a consequence, $u \in ker_{SEM}(Q \cup S)$, which shows that SEM is not cut. □

[2]Notice that it cannot be the case that $M^{i+} \subsetneq M^+$, for some i, otherwise M^+ would not be a minimal model of P.

B.2 Refers to Section 4.4

Proof. (of proposition 4.10)
"$\Rightarrow$"
Consequence of definition 4.18.
"$\Leftarrow$"
Let SEM be a ASM^h or ASM^m nonexistential semantics, i.e., there exists a normal logic program P such that $SEM(P) = \emptyset$. We build the following program P' that reveals SEM is a defective semantics.

$$\begin{aligned} a &\leftarrow not\ b \\ b &\leftarrow not\ a \\ P &\leftarrow not\ a \end{aligned}$$

$P \leftarrow not\ a$ is an abbreviation for a set of rules obtained by adding the literal $not\ a$ to the body of each rule of P; the atoms a, b do not belong to the language of P. Now $M = \{not\ a, b\}$ is a SEM model of the segment ${P'}^{\leq 1}$ because it is a stable model of the segment. Also $SEM({P'}^{>1}/\{b\}) = \emptyset$, since the set of rules $P \leftarrow not\ a$ is not solved by SEM. Thus, model M 1-segment supports no model of P', meaning SEM is defective. This reasoning is valid for any semantics of ASM^h or ASM^m families. □

Proof. (of proposition 4.11)
Let SEM be a ASM^h or ASM^m non existential semantics, it holding $SEM(P) = \emptyset$ for some normal logic program P. We build the following program P' that reveals SEM is a non cautious monotonic semantics.

$$\begin{aligned} a &\leftarrow not\ b \\ b &\leftarrow not\ a \\ Heads(P) &\leftarrow b \\ P &\leftarrow a \end{aligned}$$

$P \leftarrow a$ is an abbreviation for a set of rules obtained by adding the literal a to the body of each rule of P; the atoms a, b do not belong to the language of P; the scheme of rules $(Heads(P) \leftarrow b)$ represents a rule r per atom of $Heads(P)$, $Head(r) \in Heads(P)$. Now $M_* = \{not\ a, b\}$ is a SEM model of the segment ${P'}^{\leq 1}$ because it is a stable model of the segment. It is also the case that $SEM({P'}^{>1}/\{b\}) = \{not\ a, b\} \cup Heads(P)$, and hence $\{not\ a, b\} \cup Heads(P)$ is the only SEM model of P' – it is a stable model. We then have $Heads(P) \subseteq ker_{SEM}(P')$. It is now clear that the 2-valued interpretation $M = \{a, not\ b\} \cup Heads(P)$ is a SEM model of $P' \cup Heads(P)$ and is not a SEM model of P', which according to proposition 4.4 reveals SEM is not cautious monotonic. □

Proof. (of proposition 4.13)
The proof is immediate by resorting to the statement of proposition 4.10 and to program P' in the proof of proposition 4.11: SEM is non-existential and $b \in ker_{SEM}(P')$ while $b \notin ker_{SEM}(Rel_{P'}(b))$, hence revealing the failure of the global to local relevance property. Notice that $Rel_{P'}(b) = \{a \leftarrow not\ b,\ b \leftarrow not\ a\}$. □

Proof. (of proposition 4.14)
Let SEM be a semantics of the ASM^h or ASM^m families that is not global to local relevant, i.e. there is a normal logic program P and an atom v of the language of P such that $v \in ker_{SEM}(P)$ and $v \notin ker_{SEM}(Rel_P(v))$, where $Rel_P(v)$ is the subprogram of P that is relevant to v. Let T be the smallest layer of P such that $Rel_P(v) \subseteq P^{\leq T}$. We are going to build a program P^* where $P^{*\leq T}$ is a segment of P^* that corresponds essentially to $Rel_{P^*}(v)$ plus a set of definite rules, in the following way: for each layer $J \leq T$ of P, for each rule r in layer J such that $r \notin Rel_P(v)$, replace r by the set of rules

$$\begin{aligned}
&\{Head(r) \leftarrow Body(r), a_1^J,\\
&a_1^J \leftarrow a_2^J,\\
&a_2^J \leftarrow a_3^J,\\
&\cdots,\\
&a_{n_J-1}^J \leftarrow a_{n_J}^J,\\
&a_{n_J}^J \leftarrow\}
\end{aligned}$$

where the a_i^J do not belong to the language of P and the number of rules whose heads are in the set $\{a_1^J, a_2^J, a_3^J, \cdots, a_{n_J-1}^J, a_{n_J}^J\}$ is the adequate to put the rule $(Head(r) \leftarrow Body(r), a_1^J)$ in the layer $T+1$ of program P^*. Thus the layer T of P^* contains $Rel_P(v)$ and rules of the types $a_{n_J}^J \leftarrow$ and $a_i^J \leftarrow a_{i+1}^J$. Let $\mathcal{J}$ be the finite set of all the integers J for which the set $\{a_1^J, a_2^J, a_3^J, \cdots, a_{n_J-1}^J, a_{n_J}^J\}$ is not empty. It is thus the case that

$$M \in SEM(P) \Leftrightarrow (M \bigcup \cup_{J \in \mathcal{J}} \{a_1^J, a_2^J, a_3^J, \cdots, a_{n_J-1}^J, a_{n_J}^J\}) \in SEM(P^*)$$

and thus $v \in ker_{SEM}(P^*)$ [3]. It is also the case that (1) $Heads(P^{*>T}) \cap Atoms(P^{*\,\leq T}) = \emptyset$, because $Rel_P(v) \subseteq P^{\leq T}$ (this means that T is a segment of P^*), and (2) there is a model $N_* \in SEM(P^{*\leq T})$ for which $v \notin N_*$, otherwise it would be $v \in ker_{SEM}(Rel_P(v))$. Hence we conclude that the model N_* T-segment supports no model of P^*, which means that SEM is defective, as per definition 4.18. □

Proof. (of theorem 4.15)
Going left to right, the first equivalence is due to proposition 4.10 and the second equivalence is due to propositions 4.13 and 4.14. The logical implication is justified by propositions 4.10 and 4.11. □

Proof. (of proposition 4.16)
Let N be a stable model of a normal logic program P. Suppose that N is excessive with respect to layer T of P. Then $N^+_{\leq T} = M^+$ for some model M of $P^{\leq T}$ and $N \notin SM(P^{>T}/M^+)$. But this cannot be the case, since $\Gamma_P(N) = N$, where Γ is the Gelfond Lifschitz operator [GL88], and the performance of this operator is equivalent to the division of $P^{>T}$ by N^+, being the operation confluent no matter the order in which the atoms of N^+ are considered. Thus N is not excessive. Suppose now that N is an irregular model of P, with respect to layer T. This means there is no stable model M of $P^{\leq T}$ such that $N^+_{\leq T} = M^+$, which reveals that some atoms in $N^+_{\leq T}$ are not classically supported by N. Hence N is not a stable model, a contradiction, and thus N cannot be irregular. □

[3]Notice that by means of layered success operations, we may have $P^* = P \cup \{a_i^J \leftarrow : J \in \mathcal{J}, 1 \leq i \leq n_J\}$, and these operations do not change the semantics $SEM(P^*)$, since SEM is supposedly fair

Proof. (of proposition 4.17)
Let SEM be a ASM^h or ASM^m excessive semantics with respect to layer T of a normal logic program P. Let $M \in SEM(P^{\leq T})$ and $N \in SEM(P)$ be the models referred in definition 4.19, and $W \in SEM(P)$ be a model T-segment supported by M. We build a program Q that has a model $N_* \notin SEM(Q \cup S)$, $S \subseteq ker_{SEM}(Q)$, which according to proposition 4.6 reveals that SEM is not cut. Let $SEM(P^{\leq T}) = \{M_1, M_2, \cdots, M_t\}$. Let Q be the program

$$P^{\leq T}$$
$$u \leftarrow M_k^+$$
$$P_*^{>T}$$

where

- $P^{\leq T}$ is an abbreviation for the set of rules of segment $P^{\leq T}$;
- $u \leftarrow M_k^+$, $1 \leq k \leq t$, is a scheme of rules representing one rule per SEM model of $P^{\leq T}$, each body being the conjunction of the positive literals of the corresponding model; atom u does not belong to the language of P;
- $P_*^{>T}$ is a set of rules obtained from $P^{>T}$ in the following way: replace each literal L involving an atom of the set $Atoms(P^{>T}) \cap Atoms(P^{\leq T})$ either by u, in case $M \models L$, or else by *not* u, in case $M \not\models L$.

Now the following are true assertions:

1. Model M must T-segment support a SEM model W_* of program Q, where $W_*^+ = W^+ \cup \{u\}$. Thus $SEM(Q) \neq \emptyset$;
2. $u \in ker_{SEM}(Q)$, since any model of $P^{\leq T}$ satisfies the body of the rule $u \leftarrow M_k^+$;
3. For each model $M_k \in SEM(P^{\leq T})$, the valuations of the literals involving the atom u in $P_*^{>T}$, via the scheme $u \leftarrow M_k^+$, are the same, in the corresponding replacement positions, as the valuations with respect to the model $M \in SEM(P^{\leq T})$, of the literals of $P^{>T}$ involving $Atoms(P^{\leq T}) \cap Atoms(P^{>T})$;
4. Thus there is a model $N_* \in SEM(Q)$ of program Q such that $N_{*\leq T}^+ = N_{\leq T}^+ = M^+$, $u \in N_*^+$, and for each atom b of the language of $P^{>T}$ it is the case that $b \in N^+$, iff $b \in N_*^+$;
5. N_* does not belong to $SEM(Q \cup \{u\})$, since N does not belong to $SEM(P^{>T}/M^+)$, and the division $P^{>T}/M^+$ eliminates from $P^{>T}$ the same literals and rules that the division $P_*^{>T}/\{u\}$ eliminates from $P_*^{>T}$.

Due to items $4, 5$ and to proposition 4.6, it is clear that SEM is not cut. □

Proof. (of proposition 4.18)
"$\Rightarrow$"
Let SEM be a semantics of the ASM^m or ASM^h families, P a normal logic program, T a segment of P, and N a SEM model of P such that for no model $M \in SEM(P^{\leq T})$ do we

have $N^+_{\leq T} = M^+$, that is N is an irregular model with respect to segment T of program P. Let $SEM(P^{\leq T}) = \{M_1, M_2, \cdots, M_k\}$, where $M_i = \{a^i_1, \cdots a^i_{m_i}, not\ b^i_1, \cdots not\ b^i_{n_i}\}$, for each $1 \leq i \leq k$, the Herbrand base of $P^{\leq T}$ being, say $\{a^1_1, a^1_2, \cdots, a^1_{m_1}, b^1_1, b^1_2, \cdots, b^1_{n_1}\}$. We build a program Q whose language contains the atom v that does not belong to the language of P, such that $v \in ker_{SEM}(Rel_Q(v))$ and $v \notin ker_{SEM}(Q)$, which reveals that SEM is not local to global relevant. Let Q be the program

$$\begin{aligned}
&P\\
&c^i_j \leftarrow b^i_j\\
&d^i_j \leftarrow not\ c^i_j\\
&v \leftarrow a^i_1, \cdots, a^i_{m_i}, d^i_1, \cdots, d^i_{n_i}
\end{aligned}$$

where

- P is an abbreviation for the set of rules of program P;
- $(c^i_j \leftarrow b^i_j)$ and $(d^i_j \leftarrow not\ c^i_j)$ are schemes of rules representing individual rules for each pair of natural numbers (i, j), $1 \leq i \leq k$ and $1 \leq j \leq n_i$;
- $(v \leftarrow a^i_1, \cdots a^i_{m_i}, d^i_1, \cdots d^i_{n_i})$ is a scheme of rules representing a rule for each $1 \leq i \leq k$, that is a rule for each model in $SEM(P^{\leq T})$;
- The atoms v, c^i_j, d^i_j do not belong to the language of P.

Since the unique model of program $N^+ \cup \{c^i_j \leftarrow b^i_j,\ d^i_j \leftarrow not\ c^i_j,\ v \leftarrow a^i_1, \cdots, a^i_{m_i}, d^i_1, \cdots, d^i_{n_i}\}$ does not satisfy $Body(v \leftarrow a^i_1, \cdots, a^i_{m_i}, d^i_1, \cdots, d^i_{n_i})$, we have $v \notin ker_{SEM}(Q)$. On the other hand, all models in $SEM(P^{\leq T})$ satisfy $Body(v \leftarrow a^i_1, \cdots, a^i_{m_i}, d^i_1, \cdots, d^i_{n_i})$, by design of program Q – notice that no atom of the type c^i_j may be in the hypotheses set of a SEM model of $Rel_Q(v)$, since such a set would not be minimal [4]. Hence $v \in SEM(Rel_Q(v))$ and SEM is not local to global relevant. To see that this result is valid also for semantics of the ASM^m family, we just need to repeat the above reasoning, taking instead the program $Q = \{P,\ v \leftarrow a^i_1, \cdots, a^i_{m_i}\}$, the abbreviation and the scheme of rules having the meanings stated above.
"$\Leftarrow$"
Suppose now that SEM is a not local to global relevant semantics of the ASM^h or ASM^m families. Then there is a normal logic program P and an atom v of the language of P, such that $v \in ker_{SEM}(Rel_P(v))$ and $v \notin ker_{SEM}(P)$. There is thus a model $M \in SEM(P)$ such that $v \notin M$. Let T be the highest layer of P containing a rule of $Rel_P(v)$. Using a program transformation like the one used in the proof of proposition 4.14, we obtain a program P^* with a SEM model M_*, such that for no model $N_* \in SEM(P^{*\ \leq T})$ do we have $M^+_{*\ \leq T} = N^+_*$, which means that SEM is irregular by definition 4.20. □

B.3 Characterization of Some ASM^h and ASM^m Semantics with Respect to Existence, Relevance and Cumulativity

MH, MH$^{\mathbf{LS}}$, MH$^{\mathbf{Loop}}$

Existence: yes, since every normal logic program has a minimal model, and thus a minimal

[4]Notice that $Rel_Q(v) = P^{\leq T} \cup \{c^i_j \leftarrow b^i_j,\ d^i_j \leftarrow not\ c^i_j,\ v \leftarrow a^i_1, \cdots, a^i_{m_i}, d^i_1, \cdots, d^i_{n_i}\}$.

hypotheses model;
Global to local relevance: yes, due to existence and propositions 4.10, 4.13 and 4.14;
Local to global relevance: no, due to proposition 4.18 and example 4.7;
Cautious monotony: no, due to counter-example in example 4.4;
Cut: no, due to counter-example in example 4.4.

MH$^{\mathbf{Sustainable}}$
Existence: no; the following program P has no $MH^{Sustainable}$ models

$$\begin{gathered}a \leftarrow not\ a, not\ b, not\ v\\ b \leftarrow not\ b, not\ a, not\ u\\ u \leftarrow not\ u, not\ v, not\ a\\ v \leftarrow not\ v, not\ u, not\ b\end{gathered}$$

To see this is the case, notice that the minimal hypotheses models are $\{a, b\}$, $\{a, u\}$, $\{a, v\}$, $\{b, u\}$, $\{b, v\}$, $\{u, v\}$, each model being equal to the respective hypotheses set. Now for any of these models, condition

$$\forall_{h \in H}\ [(H \setminus \{h\}) \neq \emptyset \Rightarrow h \in WFM^u(P \cup (H \setminus \{h\}))],$$

fails, revealing the non existence of models of this program;
Global to local relevance: no, due to failure of existence and propositions 4.10, 4.13 and 4.14;
Local to global relevance: no, since $MH^{Sustainable}$ is irregular, as show by the following program, P

$$\begin{aligned}&d \leftarrow not\ a, not\ e\\ &a \leftarrow d\\ &e \leftarrow d\\ &-----1\\ &u \leftarrow not\ d\end{aligned}$$

whose $MH^{Sustainable}$ model $\{d, a, e\}$ is irregular, since its 1-segment decomposition positive part, $\{d, a, e\}$, does not correspond to any model of $P^{\leq 1}$;
Cautious monotony: no, due to counter-example in example 4.4;
Cut: no, due to counter-example in example 4.4.

MH$^{\mathbf{Sustainable}}_{\mathbf{min}}$
Existence: no, due to failure of existence for $MH^{Sustainable}$;
Global to local relevance: no, due to failure of existence and propositions 4.10, 4.13 and 4.14;
Local to global relevance: no, due to being irregular (it admits the same irregular model of the program used in the case of $MH^{Sustainable}$);
Cautious monotony: no, due to failure of existence and proposition 4.11;

Cut: no, due to the following counter-example P:

$$\begin{aligned} w &\leftarrow not\ a \\ w &\leftarrow not\ b \\ a &\leftarrow not\ b \\ b &\leftarrow not\ a, not\ u \\ u &\leftarrow w \end{aligned}$$

where $MH_{min}^{Sustainable}(P) = \{\{a, u, w\}, \{b, u, w\}\}$ and $MH_{min}^{Sustainable}(P \cup \{w\}) = \{\{a, u, w\}\}$, which reveals $MH_{min}^{Sustainable}$ as not cut due to corollary 4.7.

$\mathbf{MH^{Regular}}$
Existence: yes, since any normal logic program has a minimal hypotheses regular model. The argument is as follows. If P has a single layer, then all models of P are regular. If not, then proceed with the following iterative computation: compute a minimal hypotheses model of segment 1 of P, let it be M_1; compute a minimal hypotheses model of segment 1 of P/M_1^+; let it be M_2; repeat this procedure until a step k is reached for which $WFM^u(P/M_k^+) = \emptyset$. The T-segment decomposition $M_{k\ \leq T}^+$ with respect to any segment T of P, corresponds to a minimal hypotheses model of $P^{\leq T}$, and thus $M = WFM(P/M_k^+)$ is regular;
Global to local relevance: yes, due to existence and propositions 4.10, 4.13 and 4.14;
Local to global relevance: yes, due to $MH^{Regular}$ being regular, by design, and to proposition 4.18;
Cautious monotony: no, due to counter-example in example 4.4;
Cut: no, due to excessiveness revealed by example 4.8 and proposition 4.17.

Navy
Existence: yes, since any normal logic program has a minimal model;
Global to local relevance: yes, due to existence and propositions 4.10, 4.13 and 4.14;
Local to global relevance: no, due to irregularity, revealed by example 4.7, and to proposition 4.18;
Cautious monotony: yes, see proof below;
Cut: no, due to excessiveness revealed by example 4.6 and proposition 4.17.

Proof that $Navy$ is cautious monotonic.
Let $S \subseteq ker_{Navy}(P)$. If $M \in Navy(P \cup S)$ then $M \in Navy(P)$, otherwise there would be no minimal model of P containing S. According to corollary 4.5 this means that $Navy$ is cautious monotonic. □

Blue
Existence: yes, since any normal logic program has a minimal model;
Global to local relevance: yes, due to existence and propositions 4.10, 4.13 and 4.14;
Local to global relevance: no, due to irregularity, revealed by example 4.7, and to proposition 4.18;
Cautious monotony: yes, see proof below;

Cut: yes, see proof below.

Proof that *Blue* is cumulative.
$kernel_{Blue}(P)$ is computed by means of the following iterative procedure:

$$\begin{aligned} S_1 &= kernel_{Navy}(P) \\ S_2 &= kernel_{Navy}(P \cup S_1) \\ &\ldots \\ S_n &= kernel_{Navy}(P \cup S_{n-1}) \end{aligned}$$

until a fixpoint $S_n = S_{n+1}$ is attained. This fixpoint exists, on basis of the following argument:

1. $S_1 \subset S_2 \subset \cdots \subset S_n$.
2. P is a finite ground normal logic program, with a finite Herbrand base. Thus the sequence (S_i) must be finite.

Now we are going to show that for any $S \subseteq kernel_{Blue}(P)$ we have $Blue(P) = Blue(P \cup S)$, which together with theorem 4.9 reveals that *Blue* is a cumulative semantics.

1. Let then $S \subseteq kernel_{Blue}(P) = S_n$. There must be a $k \in \mathbb{N}_0$, $k < n$, such that $S_k \subseteq S \subseteq S_{k+1}$, where $S_0 = \emptyset$.
2. We have $Navy(P \cup S_{k+1}) \subseteq Navy(P \cup S) \subseteq Navy(P \cup S_k)$.
3. Let $S'_1 = kernel_{Navy}(P \cup S)$. We have $S_{k+1} \subseteq S'_1 \subseteq S_{k+2}$, where $S_{k+2} = S_n$ if $k + 2 \geq n$, and hence $Navy(P \cup S_{k+2}) \subseteq Navy(P \cup S'_1) \subseteq Navy(P \cup S_{k+1})$. Let $S'_2 = kernel_{Navy}(P \cup S'_1)$.
4. From the above steps we see that the sequence $S'_1, S'_2, \cdots$, is monotonically increasing (with respect to set inclusion) and finite, meaning that an index r exists such that $S'_r = S'_n$, and hence $Blue(P) = Blue(P \cup S)$. □

Cyan
Existence: yes, since any normal logic program has a minimal regular model – proof is analougous to the one given to show existence stands to $MH^{Regular}$;
Global to local relevance: yes, due to existence and propositions 4.10, 4.13 and 4.14;
Local to global relevance: yes, due to regularity, by design, and to proposition 4.18;
Cautious monotony: yes, see proof below;
Cut: yes, see proof below.

Proof that *Cyan* is cumulative.
$kernel_{Cyan}(P)$ is computed by means of the following iterative procedure:

$$\begin{aligned} S''_1 &= kernel_{Navy+Regular}(P) \text{ where } M \in (Navy + Regular)(P) \\ &\quad \Leftrightarrow M \text{ is } Regular \text{ and } M \in Navy(P) \\ S''_2 &= kernel_{Navy+Regular}(P \cup S''_1) \\ &\ldots \\ S''_t &= kernel_{Navy+Regular}(P \cup S''_{t-1}) \end{aligned}$$

until a fixpoint $S''_t = S''_{t+1}$ is attained. This fixpoint exists, on basis of an argument analogous to the one given for *Blue* semantics. The rest of the proof mimics the proof undertaken above for the *Blue* semantics. We are going to show that for any $S \subseteq kernel_{Cyan}(P)$ we have $Cyan(P) = Cyan(P \cup S)$, which by theorem 4.9 reveals that *Cyan* is a cumulative semantics.

1. Let then $S \subseteq kernel_{Cyan}(P) = S''_t$. There must be a $k \in \mathbb{N}_0$, $k < t$, such that $S''_k \subseteq S \subseteq S''_{k+1}$, where $S''_0 = \emptyset$.

2. We have $(Navy + Regular)(P \cup S''_{k+1}) \subseteq (Navy + Regular)(P \cup S) \subseteq (Navy + Regular)(P \cup S''_k)$.

3. Let $S'_1 = kernel_{Navy+Regular}(P \cup S)$. We have $S''_{k+1} \subseteq S'_1 \subseteq S''_{k+2}$, where $S''_{k+2} = S''_t$ if $k + 2 \geq t$, and hence $(Navy + Regular)(P \cup S''_{k+2}) \subseteq (Navy + Regular)(P \cup S'_1) \subseteq (Navy + Regular)(P \cup S''_{k+1})$. Let $S'_2 = kernel_{(Navy+Regular)}(P \cup S'_1)$.

4. From the above steps we see that the sequence $S''_1, S''_2, \cdots$, is monotonically increasing (with respect to set inclusion) and finite, meaning that an index r exists such that $S''_r = S''_t$, and hence $Cyan(P) = Cyan(P \cup S)$. $\square$

SM
Existence: no, due to example 4.5 and proposition 4.10;
Global to local relevance: no, due to existence failure and propositions 4.10, 4.13 and 4.14;
Local to global relevance: yes, due to propositions 4.16 and 4.18;
Cautious monotony: no, due to existence failure and proposition 4.11;
Cut: yes, see [Dix95b].

Green
Existence: yes, since any normal logic program has a minimal model;
Global to local relevance: yes, due to existence and propositions 4.10, 4.13 and 4.14;
Local to global relevance: no, due to irregularity, revealed by example 4.7, and to proposition 4.18;
Cautious monotony: no, because this semantics mimics *SM* in example 4.2;
Cut: no, due to example 4.6 and proposition 4.17.

Appendix C

Refers to Chapter 5

C.1 Refers to Section 5.2

Proof. (of proposition 5.7) We prove by mathematical induction on n, that if $b \notin \Gamma^n_{P^{t-o}}(\emptyset)$, then also $b^o \notin \Gamma^n_{P^{t-o}}(\emptyset)$.

Base Step

We show that the result is true for $n = 1$. For every rule $r \in P^{t-o}$, say ($r = b \leftarrow A, not\ B^o$), there is a co-rule $r^o \in P^{t-o}$, $r^o = (b^o \leftarrow A^o, not\ B, not\ \neg b)$. The Gelfond-Lifschitz transformation of P^{t-o} modulo $\emptyset$, $\frac{P^{t-o}}{\emptyset}$, is thus formed by pairs of co-rules, which are equal rules if we ignore the superscript "o". Thus if $b \notin \Gamma^1_{P^{t-o}}(\emptyset)$ then also $b^o \notin \Gamma^1_{P^{t-o}}(\emptyset)$.

Induction Step

Suppose (5.2) is correct for some $n = k \geq 1$. Let us show it is also correct for $n = k + 1$.

1. Assume that there is an atom $c \in \Gamma^k_{P^{t-o}}(\emptyset)$ whilst $c \notin \Gamma^{k+1}_{P^{t-o}}(\emptyset)$.

2. Then some rule in the subprogram relevant to atom c must have been eliminated by the division $\frac{P^{t-o}}{\Gamma^k_{P^{t-o}}(\emptyset)}$, whilst that rule were not eliminated by the division $\frac{P^{t-o}}{\Gamma^{k-1}_{P^{t-o}}(\emptyset)}$, otherwise we should have $c \in \Gamma^{k+1}_{P^{t-o}}(\emptyset)$. The eliminated rule must be a non superscript head one, since in definite programs $\frac{P^{t-o}}{\Gamma^i_{P^{t-o}}(\emptyset)}$, $i \in \mathbb{N}$, the subprogram relevant for a non superscript head rule contains only non superscript head rules.

3. Let $k \leftarrow X, not\ Y^o$ be one of such eliminated rules, where X (resp. Y^0) represents a conjunction of non superscript (resp. superscript) atoms.

4. This means there is an atom $y^o \in Atoms(Y^o)$ such that $y^o \in \Gamma^k_{P^{t-o}}(\emptyset)$.

5. Now due to the inductive hypothesis, we also have $y \in \Gamma^k_{P^{t-o}}(\emptyset)$. Thus the co-rule $k^o \leftarrow X^o, not\ Y$ is also eliminated by the division $\frac{P^{t-o}}{\Gamma^k_{P^{t-o}}(\emptyset)}$.

6. Hence for every rule r of P^{t-o} eliminated by the division $\frac{P^{t-o}}{\Gamma^k_{P^{t-o}}(\emptyset)}$, the co-rule r^o is also eliminated. Thus if $c \notin \Gamma^{k+1}_{P^{t-o}}(\emptyset)$ then also $c^o \notin \Gamma^{k+1}_{P^{t-o}}(\emptyset)$. This finishes the proof.

□

C.2 Refers to section 5.3

Proof. (of lemma 5.16) The system applied on Q terminates, because Q has a finite set of literals and the operations P, S eliminate literals. Let us then show its confluence. Consider two different sequences of X operations on Q, seq_1 and seq_2, of the form

$$seq_\alpha = \langle (r_1^\alpha, op_1^\alpha), (r_2^\alpha, op_2^\alpha), \cdots, (r_j^\alpha, op_j^\alpha), \cdots, (r_{n_\alpha}^\alpha, op_{n_\alpha}^\alpha) \rangle, \quad \alpha \in \{1,2\} \tag{C.1}$$

where $op_j^\alpha \in \{P, S\},\ j = 1, \cdots, n_\alpha$, and each r_j^α represents the rule of Q_j^α used to perform operation op_j^α, where Q_j^α is the program resulting from the application of the j^{th} operation in seq_α on the program Q_{j-1}^α, $j \geq 1$, being $Q_0^\alpha = Q$. Suppose the resulting programs $Q_{n_\alpha}^\alpha$, $\alpha \in \{1,2\}$, are invariant to any further application of operations of the reduction system X. We have to show that $Q_{n_1}^1 = Q_{n_2}^2$. We prove the lemma by induction on the number t of new facts produced by sequence seq_1. Recall that the operations P, S do not eliminate rules.
<u>Base step</u> If $t = 0$, that is, no new facts are produced in the n_1 steps of seq_1, then operation S uses only the set of facts of the original program. As no rules are deleted by operations P, S, the operation P uses only the atoms without a rule of the original program Q. Hence, any literal erased by a P or a S operation at some of the n_1 steps of seq_1, must also be erased by a P or a S operation at a certain step of seq_2, in order to obtain a program $Q_{n_2}^2$ invariant under any further P or S operation. As the roles of seq_1 and seq_2 may be interchanged in this reasoning, we have $Q_{n_1}^1 = Q_{n_2}^2$.
<u>Induction step</u> Suppose now that this equality is true for $t = m$. We are going to show that it is also true for $t = m + 1$. Let k be the step of seq_1 at which the $(m+1)^{th}$ new fact is generated. By the inductive hypothesis, there is some step i of seq_2 such that all the first m new facts were generated in previous $\leq i$ steps. Now the set of facts F_{k-1}^1 and the set of atoms without a rule A_{k-1}^1 in Q_{k-1}^1, are also facts and atoms without a rule in the program Q_i^2, and they constitute the preconditions for performing all the operations $\{P, S\}$ that occur in seq_1 before the $(m+1)^{th}$ fact is generated. Thus the $(m+1)^{th}$ fact must also be generated at some step $> i$ of seq_2, for the sake of invariance of the program $Q_{n_2}^2$under the system X. This finishes the proof. □

Proof. (of lemma 5.17) The rewriting system applied on Q terminates, because Q has a finite set of literals and rules, and the operations P, S, E eliminate literals and rules. Let us then show its confluence. Consider two different sequences seq_1 and seq_2 of X^* operations, of the form

$$seq_\alpha = \langle (r_1^\alpha, op_1^\alpha), (r_2^\alpha, op_2^\alpha), \cdots, (r_j^\alpha, op_j^\alpha), \cdots, (r_{n_\alpha}^\alpha, op_{n_\alpha}^\alpha) \rangle, \quad \alpha \in \{1,2\} \tag{C.2}$$

where $op_j^\alpha \in \{P, S, E\},\ j = 1, \cdots, n_\alpha$, and each r_j^α represents the set of rules of program Q_{j-1}^α used to perform operation op_j^α, Q_j^α being the resulting program, $Q_0^\alpha = Q$. Suppose the resulting programs $Q_{n_\alpha}^\alpha$, $\alpha \in \{1,2\}$, are invariant to any further application of operations of the reduction system X^*. We have to show that $Q_{n_1}^1 = Q_{n_2}^2$. We prove the lemma by induction on the number k of E operations that appear in seq_1. It is a consequence of the proof of lemma 5.16, that $k = 0$ in sequence seq_1 iff $k = 0$ in sequence seq_2 – in this case the resulting programs are equal. Let us then consider $k \neq 0$.
<u>Base Step</u>
Suppose that $k = 1$, that is, only one operation of the type E is performed in seq_1. Let us

suppose that the E operation is performed at step j of the sequence. Then all operations performed at steps $(< j)$ belong to the set $\{P, S\}$. Now none of these two operations performs rule deletion, and all the literals erased and new facts produced in the first $(j-1)$ steps of seq_1 must also be erased and produced after some step in seq_2, by the following reasoning:

- Let F, A, be respectively the set of facts and the set of atoms without a rule in the initial program Q.
- Let R be the set of nonempty body rules of Q that become facts after the first $j-1$ steps of seq_1. Then the Gelfond-Lifschitz division $\frac{R \cup F}{\emptyset}$ is a definite program, its only model satisfying the bodies of all the rules of R, and containing the facts in Q^1_{j-1}. Thus no rule of R can be deleted at any step of seq_2, and hence the facts of Q^1_{j-1} must also pertain to $Q^2_{n_2}$.

Summing it all up, the preconditions for performing an operation E at step j of seq_1, i.e. the assertion of a certain number of facts and atoms without rule, are also verified at some step of seq_2. Hence it is not possible to avoid the same operation E to be performed in seq_2, erasing thus the same set of rules (in case they were not eliminated by an alternative E operation), in order to get an invariant program $Q^2_{n_2}$ with respect to the reduction system X^*.
Induction Step
If $Q^1_{n_1} = Q^2_{n_2}$ for $k = m$, then we also have $Q^1_{n_1} = Q^2_{n_2}$ for $k = m+1$, by the following reasoning.

- Let j be the step of seq_1 at which the $(m+1)^{th}$ E operation takes place, $j-t \geq 1$ the step of seq_1 at which the m^{th} E operation takes place, and F^1_{j-t}, A^1_{j-t} be respectively the facts and the atoms without a rule in Q^1_{j-t}.
- By inductive hypothesis, there is a step $(j'-t') \geq 1$ in seq_2 where the m^{th} E operation takes place. Notice that the sequences $(F^\alpha_1, F^\alpha_2, \cdots)$ and $(A^\alpha_1, A^\alpha_2, \cdots)$, $\alpha = 1, 2$, are monotonically increasing with respect to set inclusion.
- Now from steps $(j-t)$ to $(j-1)$ in seq_1, the operations performed are P or S. By a reasoning analogous to the one used in the base step, all the new facts generated in these steps must also be generated after the step $j'-t'$ in seq_2. Thus the preconditions for performing the $(m+1)^{th}$ operation at step j of sequence seq_1 are also verified after some step, say j', of seq_2, which means that the rules eliminated after step j in seq_1 are also eliminated after step j' of seq_2.

Thus the same reductions that occur in Q to produce $Q^1_{n_1}$ also occur in Q while computing $Q^2_{n_2}$. As the roles of seq_1 and seq_2 may be interchanged in the above reasoning, we conclude that $Q^1_{n_1} = Q^2_{n_2}$. This finishes the proof. □

Proof. (of theorem 5.19) Let Q be an extended normal logic program and Q^{t-o} its $t-o$ transformed. Let seq_1 and seq_2 be two sequences of transformations of the type, $seq_\alpha = \langle op^\alpha_1, op^\alpha_2, \cdots, op^\alpha_j, \cdots,$
$op^\alpha_{n_\alpha}\rangle$, $1 \leq \alpha \leq 2$, that correspond to the application respectively of the reduction system $\mapsto_{bLWFS}$, $(\alpha = 1)$, and of the reduction system $\mapsto_{WFS}$, $(\alpha = 2)$, on the program Q^{t-o},

where the op_j^α are operations of the corresponding reduction systems. Let Q_j^α be the program obtained after the application of the transformation op_j^α on the program Q_{j-1}^α, where $Q_0^\alpha = Q^{t-o}$. Let also F_j^α, A_j^α be respectively the set of facts and the set of atoms with no rule in the program Q_j^α. We shall prove the following relations:

$$F_{n_1}^1 \subseteq F_{n_2}^2 \tag{C.3}$$
$$A_{n_1}^1 \subseteq A_{n_2}^2 \tag{C.4}$$

which, taken together, are equivalent to the relation (5.3). The proof will be by mathematical induction on the number k of bLN operations that occur in the sequence seq_1.
Base Step If $k = 0$, then no bLN operations occur in the sequence seq_1, meaning that all the operations involved belong to the set $\{P, S, F, L\}$. By the confluence of both the systems $\{P, S, F, L\}$ and $\mapsto_{WFS}$, seq_2 can be modified so that its first n_1 operations are exactly the operations of seq_1, in the same order. After these operations in seq_1 and seq_2 we have:

$$F_{n_1}^1 = F_{n_1}^2 \tag{C.5}$$
$$A_{n_1}^1 = A_{n_1}^2. \tag{C.6}$$

Now we have $Q_{n_1}^1 = Q_{n_1}^2 = bQ^{t-o}$ and seq_1 is over. If any further operation may occur in seq_2, then it must be a negative reduction operation that corresponds to no balanced layered negative reduction operation in seq_1. Being this the case, bQ^{t-o} must contain a fact and a rule whose body contains the default negation of this fact, and the operation takes place in seq_2 at step $n_1 + 1$. This makes the following relations true:

$$F_{n_1}^1 = F_{n_1+1}^2 \tag{C.7}$$
$$A_{n_1}^1 \subseteq A_{n_1+1}^2, \tag{C.8}$$

which may produce the preconditions for the application of the transformations P or F or L on $Q_{n_1+1}^2$, that may lead to S and N operations, etc. As is easily seen, the sequences $(F_1^\alpha, F_2^\alpha, \cdots)$ and $(A_1^\alpha, A_2^\alpha, \cdots)$ are monotonically increasing with respect to set inclusion, and thus:

$$F_{n_1}^1 \subseteq F_{n_s}^2 \tag{C.9}$$
$$A_{n_1}^1 \subseteq A_{n_s}^2, \tag{C.10}$$

where n_s indexes the last step of seq_2. The relations C.3 and C.4 are thus valid for the case $k = 0$.
Induction Step We now show that if relations C.3 and C.4 are true for $k = m$ then they are also true for $k = m + 1$. Suppose then that the relations are valid for $k = m$. Let i (resp. $i - t$) be the index of the step in seq_1 at which the $(m + 1)^{th}$ (resp. m^{th}) bLN operation takes place. By the confluence of the system $\mapsto_{WFS}$, we can make seq_1 and seq_2 equal in the first $i - t$ steps (this may correspond to postpone some negative reduction operations in the $\widehat{Q^{t-o}}$ computation in seq_2) and then

$$F_{i-t}^1 = F_{i-t}^2 \tag{C.11}$$
$$A_{i-t}^1 = A_{i-t}^2 \tag{C.12}$$
$$Q_{i-t}^1 = Q_{i-t}^2. \tag{C.13}$$

Now all the operations performed on program Q^1_{i-1} from steps $i-t$ to $i-1$ in sequence seq_1, belong to the set $\{P, S, F, L\}$, and can also be performed on Q^2_{i-1}, in seq_2. Thus we can write

$$F^1_{i-1} = F^2_{i-1} \tag{C.14}$$
$$A^1_{i-1} = A^2_{i-1} \tag{C.15}$$
$$Q^1_{i-1} = Q^2_{i-1}. \tag{C.16}$$

and the preconditions for the application of the $(m+1)^{th}$ bLN operation are verified by both Q^1_{i-1} and Q^2_{i-1}. Hence,

$$F^1_i = F^2_i \tag{C.17}$$
$$A^1_i = A^2_i \tag{C.18}$$
$$Q^1_i = Q^2_i. \tag{C.19}$$

The validity of relations C.3 and C.4 for the case $k = m+1$ is immediate by continuing this reasoning in a manner analogous to the one used in the base step. □

C.3 Refers to section 5.4

Proof. (of theorem 5.21) Let P be a normal logic program. We start by proving the equalities

$$bP^{t-o} = \overset{o}{P^{t-o}} \tag{C.20}$$
$$\overset{o}{P}{}^{t-o} = \overset{o}{P^{t-o}}, \tag{C.21}$$

where (C.20) means that the balanced layered remainder bP^{t-o} is equal to the layered remainder $\overset{o}{P^{t-o}}$ of P^{t-o}, and (C.21) means that the $t-o$ transformed $\overset{o}{P}{}^{t-o}$ of $\mathring{P}$, is equal to the layered remainder $\overset{o}{P^{t-o}}$ of P^{t-o}.

1. (proof of C.20)

   (a) Let $P^{t-o}_{\neg}$ be the program that results from applying to P^{t-o} the operations of positive reduction necessary to eliminate all default negations of explicitly negated literals, say *not* $\neg b$ (notice that no explicitly negated literal has a rule in P^{t-o}). Due to the confluence of the system $\mapsto_{bLWFS}$ (resp. $\mapsto_{LWFS}$), any sequence of operations for computing bP^{t-o} (resp. $\overset{o}{P^{t-o}}$) may take all these positive reduction operations as the first set of operations performed. Thus the computation of bP^{t-o} (resp. $\overset{o}{P^{t-o}}$) may be performed in a manner such that the program $P^{t-o}_{\neg}$ appears at a certain step of the computation. Each of the sets $(\triangle P^{t-o}_{\neg})^+$, $(\triangle P^{t-o}_{\neg})^u$, $(\triangle P^{t-o}_{\neg})^-$, contains only pairs of co-atoms, by definition 5.3 (of $t-o$ transformation).

(b) Let *op* be the first operation performed on $P^{t-o}_{\neg}$ in the sequel of the computation of $\overset{o}{P}^{t-o}$. Then *op* either deletes (i) pairs of co-atoms (success), or (ii) pairs of co-atoms default negations (positive reduction), or (iii) sets of co-rules (loop detection, failure, layered negative reduction), by point (1a) above (obs. rules in $P^{t-o}_{\neg}$ appear as pairs of co-rules). The same happens if *op* is the first operation performed in the sequel of the computation of bP^{t-o}. Notice that balanced negative reduction and layered negative reduction are equivalent in this context, since a rule r is involved in a loop through say the literal *not* c^o iff its co-rule r^o is involved in a loop through the literal *not* c, as per observation and point (1a) above. Thus, the program P^{t-o}_1 resulting after this first operation is the same in both cases.

(c) Point (1b) above constitutes the base step of an inductive proof of equality (C.20), where the induction is on the number of operations "op" performed, whose inductive step proceeds in an analogous vein, showing that if P^{t-o}_i is the program resulting after the i^{th} reduction operation, then $(\triangle P^{t-o}_i)^+$, $(\triangle P^{t-o}_i)^u$, $(\triangle P^{t-o}_i)^-$ contain only pairs of co-atoms, which in turn shows that the systems $\mapsto_{bLWFS}$ and $\mapsto_{LWFS}$ produce the same resulting program when applied on the $t-o$ transformed of a normal logic program P.

2. (proof of C.21)

(a) Let $P^{t-o}_{\neg}$ be the program obtained in point (1a) above. The following equalities prevail, by definition 5.3:

$$b \in (\triangle P)^+ \text{ iff } b, b^o \in (\triangle P^{t-o}_{\neg})^+ \tag{C.22}$$

$$b \in (\triangle P)^- \text{ iff } b, b^o \in (\triangle P^{t-o}_{\neg})^- \tag{C.23}$$

$$b \in (\triangle P)^u \text{ iff } b, b^o \in (\triangle P^{t-o}_{\neg})^u. \tag{C.24}$$

Thus for each operation of the system $\mapsto_{LWFS}$ performed on P (or on the program that results after performing on P a set of operations of this system), that creates a new fact c, eliminates a literal *not* c, or eliminates a set of rules R, there is a corresponding set of operations of the system $\mapsto_{bLWFS}$ performed on $P^{t-o}_{\neg}$ (or on the program that results after performing on $P^{t-o}_{\neg}$ a set of operations of this system), that respectively creates the new facts c, c^o, eliminates the pair of literals *not* c, *not* c^o, or eliminates the set of rules R^{t-o}, where $r \in R$ iff s, $s^o \in R^{t-o}$, s, s^o resulting from r by $t-o$ transformation.

(b) The previous point, together with definition 5.3, shows the validity of equality (C.21).

Now from the equalities (C.20) and (C.21) we have

$$bP^{t-o} = \overset{o}{P}^{t-o}. \tag{C.25}$$

This means that the interpretations $\triangle bP^{t-o}$ and $\triangle \overset{o}{P}^{t-o}$ are equal, and have thus the same set of pairs of undefined co-atoms. Hence the set of MH_P assumable hypotheses of P is also the set of MH assumable hypotheses of P. If $Hyps(P)$ is the set of MH_P hypotheses of P and $H \subseteq Hyps(P)$, then we have,

$$\begin{aligned} WFM_P(P \cup H) &= \triangledown WFM(P^{t-o} \cup H \cup \{h^o \leftarrow \mathit{not}\ \neg h : h \in H\}) \\ &= \triangledown WFM(bP^{t-o} \cup H \cup\ \{h^o \leftarrow \mathit{not}\ \neg h : h \in H\}) \\ &= \triangledown WFM(\overset{o}{P}{}^{t-o} \cup H \cup\ \{h^o \leftarrow \mathit{not}\ \neg h : h \in H\}) \\ &= WFM_P(\overset{o}{P} \cup H), \end{aligned}$$

and hence

$$WFM_P(P \cup H) = WFM_P(\overset{o}{P} \cup H).$$

Now this equality and theorem 5.4 immediately show that if P is a normal logic program, then the MH and MH_P models coincide, if we ignore the default negations of explicitly negated literals in the MH_P models, introduced by the language of the $t - o$ transformed, because by doing so the models $WFM_P(P \cup H)$ and $WFM(P \cup H)$ are the same. $\square$

Proof. (of theorem 5.24) Let us show that finding a MH_P model of an extended normal logic program P is a Σ_2^P-complete task. Computing P^{t-o} is fulfilled in linear time. The balanced layered remainder bP^{t-o} is computed in polynomial time, by the following reasoning: the calculus of the remainder of a normal logic program is known to be of polynomial time complexity [BDFZ01]; the difference between $\mapsto_{WFS}$ and $\mapsto_{bLWFS}$ lies on the operator $\mapsto_N$ of the former being replaced by the operator $\mapsto_{bLN}$ of the latter; to perform $\mapsto_{bLN}$ the rule layering must be computed; the rule layering can be calculated in polynomial time since it is equivalent to identifying the *strongly connected components* [Tar72] in a graph, *SCCs*, in this case in the complete rule graph of P^{t-o}; when verifying the preconditions to perform a negative reduction operation (existence of a fact, say b, and a rule with *not* b in the body), it is linear time to check if a rule is in loop (check if it belongs to a SCC) and if it is in loop through literal *not* b (check if b belongs to the heads of the SCC) – the same for checking if the co-rule is in loop through *not* b^o; therefore, balanced layered negative reduction adds only polynomial time complexity operations over negative reduction. Once bP^{t-o} is computed, non deterministically guess a set H of hypotheses – the assumable hypotheses set is the set of all superscript atoms involved in default negations in bP^{t-o}, whose co-atoms are not facts. Check if $WFM_P^u(P \cup H) = \emptyset$ – this is polynomial time. Checking that H is empty or non-empty minimal, requires another non deterministic guess of a strict subset H' of H and then a polynomial check if $WFM_P^u(P \cup H') = \emptyset$. $\square$

Index