Centro de Inteligência Artificial
Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

Revising Undefinedness
in the Well-Founded Semantics of Logic Programs

Luís Filipe Rodrigues Soares

A thesis submitted to the Department of Computer Science and the Committee on Graduate Studies of Universidade Nova de Lisboa in partial fulfillment of the requirements for the degree of Master of Science

Caparica, 2006

This dissertation was prepared under the supervision of
Professor Luís Moniz Pereira,
of the Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa.

to the dropplets of
water,
bashing in the
haul
of an intense vessel,
fading
as life goes on;
such are the
difficulties and
fears.

[This page was intentionally left blank]

Acknowledgements

This thesis, completed within a year of very intense experiences, actually spans over the last three or four years of my life. Those were truly crucial years in setting the motivation for coming to Nova and persuing this line of work, now giving its first results. For this, I thank Isa. During all these years she always supported me and believed me, and without her company and encouragement this work wouldn't have happened. She was always available to listen to my wandering thoughts on logic and absurdity, as well as my concerns and fears regarding my work and its success. The moments I spent with her were always beautiful and full of joy and tranquility. Her love and friendship were the best strenghts I could have had to give this first step, and those are things that I *always* keep with me. Thanks.

My mother and my sister are among the most patient persons in the world. They always let me do things my way, even when it interfered with their lives and schedules, and they always beared with my sudden special needs, even when this meant more work for them. My mother was specially inspiring in the way she has always fought everything without ever giving up, a beautiful persistence I'm sure I got from her.

On the academic side, Professor Moniz Pereira made this whole adventure possible. He was the kind of supervisor I wanted, always keeping me occupied with challenges and always available for my doubts and difficulties. The fact that he accepted me as his student and that he believed in my work was of the utmost importance to me. His constant dedication to scientific research is truly inspiring and his contagious sense of humor was essential for the completion of this thesis.

Thanks are also due to the Departamento de Informática and the Centro de Inteligência Artificial, for providing me with working conditions and great examples among the people working there. The Secretariat staff has always been helpful and patient in all the things I required, for which I also thank them. A special word of thanks also goes to the Multimedia and Databases Laboratories research group, at Escola Superior de Tecnologia, Instituto Politécnico de Setúbal, for providing me alternative working conditions.

Sérgio, Carmen, Alexandre, Miguel, Hugo Silva and João Leite were of great help in proofreading my thesis, for which I thank them.

I also had the support of the best working partners and friends a guy could have. Alex was

always helpful with insightful opinions and a few extra minutes to spend listening to my latest ideas. He was a great help in all the hints and suggestions he gave to improve my work. And there is also Sérgio, who was always there to listen to me talking about my work, even though he had his own concerns with pixels, isosurfaces and volume rendering. He was one of the few people who actually understood this work, contributed with new problems and examples, even spent some time solving some of these issues and yet this had nothing to do with what he was working on. We spent some great relaxing moments in math's bar, just talking and enjoying the *views*. Those *views* were indeed crucial to our mental sanity. Thanks for being a great friend and formidable colleague.

Doca de Santo at Alcântara and Bar Proibido at Belém were alternative working places. Besides having good working environments, these were also relaxing places to escape from work.

João Leite was also of great help and inspiration ever since I came to Nova. These ideas weren't among his favourites and so our discussions were a great test to my persistence and my belief in my work, which I think only made it stronger. Besides, he was a great professor, a great research partner and an example both in achievements and dedication to his work.

A special thanks also goes to Sophie, who opened my eyes to a lot of important issues during this period of time. Our moments were filled with brevity but set many important things in motion regarding me and this thesis. We ended up sharing *more* than just music, which had a great impact on me and my work.

Miguel, Mité and Carmen were the greatest friends I could have had. Their companion, support and advices were more than essential to me. Carmen has accompanied me since I came to Nova and has always given me important advices, as well as great moments of fun, along with her big Vimi. Mité was a fun companion in all the singing and guitar playing in her house and at Amadeu's. Not everyone could appreciate the greatness of our voices, which only means they are *that* special. Miguel has probably interfered more with my life than any other person. He was my professor, my supervisor, my work partner and is something like a best friend and an older brother to me. He always had time and the moments I spent in his company were always enjoyable and relaxing. Besides all this, he has always been an inspiration in so many ways. Thank you for always being there.

Finally, there is Maria, who has (a)mused me when I needed it the most, and gave me so many crazy and passionate moments; an intensity I was missing and which I truly needed. Thanks.

Summary

The Well-Founded Semantics (WFS) for normal logic programs associates with each program one single model expressing truth, falsity and undefinedness of atoms. Under the WFS, atoms are said to be undefined if:

- Either are part of a two-valued choice (true in some worlds, false in others) but never undeniably true or false;
- Or depend on an already undefined literal;
- Or are not classically supported.

Undefinedness due to lack of classical support could be overcome by the introduction of another form of support, which would allow the WFS to correctly deal with programs requiring non-classical forms of reasoning, and thus gain expressiveness. One of these forms of *unclassical* support whose application has already been studied in the Revised Stable Models semantics (rSMs) is support by *reductio ad absurdum* (RAA). This principle states that an hypothesis should be true if by assuming it's false this assumption leads to a contradition.

In this thesis we propose to study the application of the RAA principle to the WFS, thus defining the Revised Well-Founded Semantics (RWFS). Besides this definition we'll also study the definition of a fixed-point operator Γ^r, a counterpart of Gelfond-Lifschitz operator Γ, with support for RAA reasoning, and use this operator to perform the calculation of rSMs and the revised well-founded model of normal logic programs. We will also study a new property of rSMs and the definition of the revised partial stable models.

This thesis concludes with the discussion of several open issues and possible next research paths.

[This page was intentionally left blank]

Sumário

A Semântica Bem-Fundada (SBF) dos programas lógicos normais associa a cada programa um único modelo que expressa a verdade, falsidade e indefinição dos seus átomos. Os átomos, na SBF, dizem-se indefinidos se:

- Ou são parte de uma escolha a dois valores (verdadeiros em alguns mundos, falsos noutros) mas nunca são inquestionavelmente verdadeiros ou falsos;
- Ou dependem em átomos que já são indefinidos;
- Ou não têm suporte clássico.

A indefinição devida à falta de suporte clássico pode ser ultrapassada através da introdução de outra forma de suporte, o que permitiria à SBF lidar correctamente com programas que necessitassem de formas de raciocínio não-clássicas, ganhando assim expressividade. Uma destas formas de raciocínio não-clássico cuja applicação foi já estudada na semântica dos Modelos Estáveis Revistos (MERs) é o suporte por *redução ao absurdo* (RAA). Este princípio diz que uma hipótese deverá ser verdadeira se, por causa de se assumir que ela é falsa, essa assumpção nos leva a uma contradição.

Nesta tese propômo-nos a estudar a aplicação do princípio de RAA na SBF, definindo assim a Semântica Bem-Fundada Revista. Para além desta definição, vamos também estudar a definição de um operador de ponto fixo Γ^r, que será a versão com suporte por RAA do operador Gelfond-Lifschitz Γ, e vamos usar este operador para calcular os MERs e o modelo bem fundado revisto de programas lógicos normais. Vamos ainda estudar uma nova propriedade dos MERs e a definição dos modelos estáveis revistos parciais.

Esta tese culminará com uma discussão de vários pontos em aberto e do trabalho futuro a realizar neste contexto.

[This page was intentionally left blank]

Contents

Chapter 1

Introduction

Contents

Where we briefly present the motivational setting for this thesis, the problems we intend to address and the main contributions of this work. In this chapter we also outline the structure and contents of each chapter.

I know, I know for sure,
That life is beautiful around the world (...)

Red Hot Chili Peppers. Around the World. In *Californication*. Warner Bros., 1999.

1.1 Motivation

The concerns of what was known for sure were among the things that puzzled Anthony Kiedis of the Red Hot Chili Peppers, whether these concerns were with how life was around the world or what to say to a girl when she says "Hello..." [27]. To know these things for sure would imply that he had analyzed every relevant fact and rule regarding the beauty of life around the world (and conversations with girls) and, given one single observation of the world, he would be able to conclude what was true or false, and what was impossible to say if it was true or false. This single view would be skeptical, cautionate in the way it attributed truth or falsity to things. And for those cases where things weren't absolutely true or false, this view would say that things should be undefined.

Undefinedness helps enforcing a skeptical view of the world. It's a truth value used to refer to things whose truthness can't be determined. Things fall in this category, for example, when there are situations in which a certain thing is true, and other situations where the same thing is false. Things might also fall in this category when, for some reason the truth value of something just can't be determined or things depend on already undefined things. In either case it's more cautious to say that things are undefined than to compromise with truth or falsity.

The reasoning behind this idea was captured under a precise semantics for logic programs, around 1990. The Well-Founded Semantics [40](WFS) has several desirable properties as a semantics for logic programs, namely cumulativity, relevancy and existence. Furthermore, it ensures uniqueness of model, its well-founded model is computable in polynomial time and several WFS implementations ([35, 37, 39, 41]) have been studied and exist nowadays. Moreover, several important relations exist between the well-founded semantics and nonmonotonic reasoning formalisms [5, 12], as well as between the well-founded semantics and the stable models semantics [10, 34], another important semantics for nonmonotonic reasoning. Additionally, the WFS was later expanded to its explicit [2,28], paraconsistent [9] and disjunctive [1] counterparts.

Under the well-founded semantics, atoms can be true, false or undefined. They are said to be true if and only if they are undoubtfully true, i.e., they have classical support and when

transitioning to a two-valued context with several (possible) models, they are true in all models. Literals are said to be undefined if they fall in one of the following cases:

- Either the literals are part of a two-valued choice (true in some worlds, false in others) but never undeniably true or false;
- Or the literals are not classically supported;
- Or the literals depend on an already undefined literal.

Otherwise, they are false.

It's easy to see that literals that are part of choices should be undefined – choices are made in a two-valued context so things are either true or false. In a three-valued context, as they can't be true and false at the same time they remain undefined. Additionally, if a literal depends on an already undefined literal it should also be undefined[1] – undefinedness propagates throughout the program. However, lack of classical support could be overcome by the introduction of another form of support. This will be the theme of this thesis.

1.2 Approach

As we stated before, it's easy to see that literals that are part of choices should be undefined, as well as those that are dependent on atoms which already are undefined. Lack of classical support, however, could be overcome by the introduction of another form of support. By enabling other forms of support, the WFS would gain expressiveness and the ability to correctly deal with programs which required non-classical forms of reasoning. One such program is the following:

$$P = \{go\_out\_with\_me \leftarrow sad$$
$$sad \leftarrow not\ go\_out\_with\_me\}$$

This program represents one possible reasoning behind asking a girl out on a date. What the program is stating is that she should go out with me[2] if she is sad. Furthermore, she will be sad if doesn't go out with me. Under this scenario, and given that the girl in question doesn't want to be sad, she could start by questioning herself, for instance, what would happen if she refused the invitation. By assuming the hypothesis *not go_out_with_me*, she should be sad. However, being sad is precisely the precondition for "going out with me", contradicting the original hypothesis. Therefore, she couldn't refuse the invitation – she would "go out with me".

[1] Although it can be argued that, in some cases, some conclusions may be derived from literals which already were undefined, namely when we are referring to choices. We dedicate some attention to this possibility in the conclusions.

[2] Without loss of generality.

The reasoning behind this program is not a classical one. The literal *go_out_with_me* is actually concluded by reasoning by absurdity, a form of *unclassical* support. However, as the well-founded semantics doesn't employ this kind of support, both atoms $\{go\_out\_with\_me, sad\}$ would be undefined in the well-founded model. Let's consider another example, still in the same context:

$$\begin{aligned} P = \{ & go\_out\_with\_me \leftarrow sad \\ & sad \leftarrow not\ go\_out\_with\_me,\ not\ get\_drunk \\ & get\_drunk \leftarrow bored \\ & bored \leftarrow not\ get\_drunk\} \end{aligned}$$

Under this program, we are stating that the previous girl will be sad not only if she doesn't go out with me, but also if she doesn't get drunk. Regarding the ingestion of alcoholic drinks, she will usually get drunk if she is bored. However, she will get bored if she is not drunk. Following the reasoning behind the previous example, this girl will always be drunk. This state will prevent her from being sad and, therefore, of going out with me. Once again, under the well-founded semantics, all atoms will be undefined. However, if we were able to reason by absurdity in the WFS, we would conclude $drunk$, while everything else would be false.

This *reasoning by absurdity* follows the principle of *reductio ad absurdum* (RAA), which states that an hypothesis should be true if by assuming it is false this assumption leads to a contradiction. In that case, our conclusion is that our hypothesis should be true.

The application of this principle has already been studied in the Stable Models Semantics [14] (SMs) and enabled the Revised Stable Models [31, 32] (rSMs) to solve the problems associated with the SMs, while also enjoying the advantages associated with them. It also provided with an initial proof that the principle of RAA could effectively be used as a form of *unclassical* support.

Studying the application of this principle in the well-founded semantics will be the subject of this thesis.

1.3 Contributions and Outline

In this thesis we propose to study and implement the application of the *reductio ad absurdum* principle to the well-founded semantics, with the purpose of contributing with more expressive results in a three-valued context, by recurring to undefined only when dealing with choices or with propagation. The study of the application of the RAA principle in the WFS will also provide a way of transitioning between a three-valued semantics with RAA support and the the revised stable models, these also sharing the same form of *unclassical* support. We will dub this semantics the Revised Well-Founded Semantics (rWFS) and study the definition and calculation of its Revised Well-Founded Model (rWFM).

The introduction of this semantics further provides the following contributions:

- The study of the application of the principle of *reductio ad absurdum* in a three-valued scenario, as well as the implications this introduction would have when defining a semantics based in this principle;
- The definition of the principle of *reductio ad absurdum* in a three-valued context, which is an extension of the original principle, only valid in a two-valued context;
- The definition of the revised partial stable models, a complete counterpart of the partial stable models, where the path between the rWFM and the rSMs always exists and is only the result of choices on undefined literals;
- The study of RAA patterns and the consequent definition of a fixed-point operator Γ^r which is the counterpart of the Gelfond-Lifschitz operator Γ, with support for RAA reasoning;
- The definition of the revised stable models semantics and the revised well-founded semantics through the use of Γ^r operator, and other further explorations in the revised stable models semantics.

This thesis is organized as follows: Chapter 2 presents the historical and theoretical settings of this work and discusses the achievements of the three most relevant semantics in the context of this thesis. It ends with an account of the role played by each semantics and what role the rWFS should play.

In Chapter 3 we define the revised well-founded semantics. We start with an extensive motivation for the introduction of this semantics and we define the principle of RAA in a three-valued setting. We then provide several preliminary definitions which will enable us to contextualize the RAA principle and present the declarative definition of the rWFS. We then study its properties and present some examples of the calculation of the revised well-founded model.

In chapter 4 we start with a new property of the revised stable models, RAA rule extension, which will enable us to define a syntactic transformation for rSMs and the rWFS. After defining this transformation we present some programs and the calculation of their revised stable models and revised well-founded model.

This thesis ends with a discussion of some open issues and concluding remarks.

[This page was intentionally left blank]

Chapter 2

History and Theoretical Foundations

Contents

Where we present the historical and theoretical context for this thesis. We briefly present the sequence of events which resulted in this thread of research, and present the main definitions and base concepts needed to understand this work.

2.1 A Historical Overview

The field of Artificial Intelligence (AI) has grown in both depth and breadth since its very name was coined at the 1956 Dartmouth College meeting [22]. If, in those early days, AI was hardly considered a science, it didn't take long for formal methods of research and development to appear. The area soon became richer in its various approaches to the challenge of providing computers with methodologies similar to those used by humans to solve problems.

From the several approaches that followed the Dartmouth meeting, one was related with the use of logical formulas to express knowledge. It was around 1960 that John McCarthy first expressed the advantages of such approach [20]:

> "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."

Other authors defended this orientation, namely Patrick Hayes [21], Nils J. Nilsson [26] and Drew McDermott [23], which ended up originating two main threads of research, initially very distinct but later converging on the same line of work.

The first of these was Logic Programming (LP). Logic Programming, the application of automatic methods of deduction combined with logic-based knowledge representation, was started by Alain Colmerauer [7] and Robert Kowalski [16,17], and introduced in Computer Science the idea of a declarative approach to programming, as opposed to a procedural one. This new programming paradigm was summarized by Robert Kowalski in [16], and ended up providing a fast proliferation of the logic-based approach to AI, as researchers now had a way to implement and test their theories. Logic programming eventually came to life under the Prolog[1] language in 1972 by the hand of Colmerauer and Kowalski, which not only did it demonstrate that Prolog was a practical language as it also helped disseminate it worldwide. Nowadays several Prolog distributions exist, making LP a relevant paradigm in Computer Science.

During the same years, other researchers were delving into the problems of performing commonsense reasoning in AI. John McCarthy was perhaps the first person to consider these issues [20] and later, together with Patrick Hayes [21], introduced the Frame Problem. The Frame Problem deals with the representation of things that do not change in a given situation, when a certain event occurs. This frame problem was precisely the kind of problem to be addressed by nonmonotonic reasoning, the fact that monotonic classical logic *per se* does not deal correctly with changing worlds. The term *nonmonotonic reasoning* is probably attributed to Marvin Minsky in [24], where he also summarizes his critique to classical logic.

[1] Prolog is an abbreviation of *PROgrammation en LOGique*, name suggested by Philippe Roussel [8].

Both threads of research continued to evolve rather separated from each other, until 1986. In 1986, at the Workshop on the Foundations of Deductive Databases and Logic Programming, a major milestone was reached with the Stratified Semantics – it is a semantics of interest for the LP community, but its underlying principles are more closely related to those of NMR. It was a milestone precisely because this and the group of semantics that followed, left the operational concerns of classical LP behind and were more concerned with their suitability for practical applications and NMR.

During the following years, several different semantics appeared and important relations between them and NMR were discovered. Of particular interest to this work are the following moments:

- 1988, introduction of the Stable Models Semantics [14];
- 1991, introduction of the Well-Founded Semantics [40];
- 2004, introduction of the Revised Stable Models Semantics [31, 32].

These semantics set the grounds for this work and constitute the last steps in the thread of research we have described so far.

In the remainder of the chapter, we'll introduce the theoretical notions required for the understanding of this work, as well as the semantics it is based on. We end up with a motivation for the revised well-founded semantics, the main theme of this thesis.

2.2 Language of Logic Programs and Other Common Definitions

We now present the theoretical context for this thesis by showing several basic definitions that will be used throughout this work.

We start by defining an alfabet[2] $\mathcal{A}$ over a language $\mathcal{L}$, which is a finite or countably infinite disjoint set of constants, predicate symbols with associated arity, function symbols with associated arity, as well as a countably infinite set of distinguished variables.

Definition 1 (Term)**.** *A term over $\mathcal{A}$ is defined as one of the following:*

- *A variable, which we'll normally write as X, in uppercase;*
- *A constant, which we'll normally write as x, in lowercase;*
- *A function symbol $f(t_1, \ldots, t_n)$ with $t_1, \ldots, t_n$ being terms;*

A term is said to be ground if it doesn't contain variables.

[2]The definitions in this section are borrowed and extended from [2, 4, 18].

Definition 2 (Atom). *An atom is either a term or an expression $p(t_1,\ldots,t_n)$, where p is a predicate symbol and $t_1,\ldots,t_n$ are terms. An atom is ground if all its terms are grounded.*

Definition 3 (Herbrand base). *The set of all ground atoms is called the Herbrand base and is represented as $\mathcal{H}$.*

Definition 4 (Literal). *A literal is either an atom A or its default negation $\sim A$. We will use the symbol $\sim$ to refer to the implicit negation, or negation as finite failure to prove a certain literal, or simply default negation. Hence, literals can also be called positive literals or default (negative) literals, whether they are respectively non-negated literals or negated literals. A literal is said to be ground if it doesn't contain any variable.*

Definition 5 (Rule). *A rule is a clause over $\mathcal{A}$ in the form*

$$H \leftarrow A_1,\ldots,A_n$$

where H is a positive literal and $\{A_1,\ldots,A_n\}$ are positive or negative literals. H is called the head of the rule while $\{A_1,\ldots,A_n\}$ constitutes the body of the rule. If a rule doesn't contain a body it is called a fact and is represented as

$$H \leftarrow$$

Definition 6 (Heads and Bodies of rules). *Regarding rules we define the functions $head(r)$ and $body(r)$ which, for any rule r respectively give as a result its head and its body.*

Given a set of rules S, the functions $heads(S)$ and $bodies(S)$ respectively give as a result the set of all heads of all rules of S and the set of all bodies of all rules of S.

From these definitions we can now specify what we mean by a Definite Program and a Normal Logic Program.

Definition 7 (Definite Program). *A definite program (DP) P is a countable set of rules, such that their bodies only include positive literals.*

Definition 8 (Normal Logic Program). *A normal logic program (NLP) P is a countable set of rules.*

The transition from NLPs, which are only syntactic descriptions of some world or situation, to the meaning they have is done through the following definitions regarding interpretations and models.

Definition 9 (Two valued Interpretation). *A two-valued interpretation I of a NLP P is any subset of the Herbrand base $\mathcal{H}$ of P, and can be viewed as the set*

$$I = \mathcal{T} \cup \sim \mathcal{F}$$

where $\mathcal{T} \subseteq I$ is the set of atoms true in I and $\mathcal{F} = \mathcal{H} \setminus \mathcal{T}$ is the set of atoms false in I.

Definition 10 (Three valued Interpretation). *A three-valued interpretation I of a NLP P is a set*

$$I = \mathcal{T} \cup \sim \mathcal{F}$$

where $\mathcal{T}$ and $\mathcal{F}$ are disjoint sets of the Herbrand base $\mathcal{H}$ of P and $\mathcal{T}$ is the set of atoms true in I and $\mathcal{F}$ is the set of atoms false in I. The truth value of the remaining atoms is undefined.

For the sake of simplicity, we'll often refer to two-valued interpretations only by including the set of literals that are true ($\mathcal{T}$), and three-valued interpretation by including the set of literals which are true, and those which are true or undefined ($\langle \mathcal{T}, \mathcal{T}\,\mathcal{U} \rangle$).

Interpretations can be viewed as *possible worlds* of a program P, as it is discussed by Przymusinska and Przymusinski [33] which correspond to possible states of knowledge regarding program P. Three valued interpretations account for the possibility of not knowing everything about the world, and being able to represent the things known to neither be definitely true nor definitely false.

Definition 11 (Satisfaction)**.** *Given an interpretation I (two valued or three valued) and a NLP P we say that:*

- *I satisfies a literal X, denoted by $I \models X$, iff $X \in I$;*
- *I satisfies the conjunction $X_1, \ldots, X_n$, denoted by $I \models \{X_1, \ldots, X_n\}$, iff $I \models X_1, \ldots, I \models X_n$;*
- *I satisfies the rule $H \leftarrow X_1, \ldots, X_n$, denoted by $I \models (X_1, \ldots, X_n)$, iff whenever the conjunction $X_1, \ldots, X_n$ is satisfied by I, then H is also satisfied by I.*

2.3 Stable Models Semantics and the Well-Founded Semantics

Gelfond and Lifschitz introduced the Stable Models Semantics (SMs) in 1988 [14], a semantics which would end up becoming one of the most important in the field of logic programming.

The main idea behind the stable models comes from the field of nonmonotonic reasoning, following the sequence of events detailed in the previous section. Its intuition is that if we assume some literals as true and others as false, those assumptions should be corroborated by the semantics of definite programs [38]. If they are, then they form a stable model. This semantics makes use of an operator that, given a two-valued interpretation I, transforms the NLP into a definite program (DP), in order to calculate its least model.

Definition 12 (Gelfond-Lifschitz operator Γ)**.** *Let P be a NLP and I a two-valued interpretation. The GL-transformation of P modulo I is the program $\frac{P}{I}$ obtained from P by performing the following operations:*

- *Remove from P all rules containing a default literal $\sim X$, such that $X \in I$;*
- *Remove from the remaining rules all the other default literals.*

Since $\frac{P}{I}$ is a DP, it has a unique least model J.

Let $\Gamma_P(I) = J$.

Definition 13 (Stable Models Semantics)**.** *A two-valued interpretation I of a NLP P is a Stable Model of P iff* $\Gamma_P(I) = I$.

An atom X of P is true under the stable models semantics iff X belongs to all stable models of P.

We represent a stable model M of a NLP P as $SM_P(M)$.

It was soon discovered [6, 13] that the stable models are closely related to nonmonotonic reasoning formalisms, namely Reiter's default extensions [36] and Moore's autoepistemic extensions [25]. This close relation helped bring the fields of Nonmonotonic Reasoning and Logic Programming closer, by providing LP with nonmonotonic reasoning formalisms and NMR with a practical vehicle of experimentation and research.

Besides this important relation, the SMs ended up being defined for the extended class of logic programs, i.e., logic programs with classical negation. The resulting semantics, the Answer Set Semantics [15] is one of the most used in Multiagent Systems and Artificial Intelligence applications where the knowledge representation approach based on logic.

The SMs have, however, some important drawbacks. First of all, the stable models semantics is not always consistent, which means that some programs have no semantics at all. Consider, for example

$$P = \{a \leftarrow\sim a\}$$

$\Gamma_P(\{a\}) = \{\}$ but $\Gamma_P(\{\}) = \{a\}$, which means P has no semantics whatsoever, because there is no single interpretation $I \subseteq \mathcal{H}_P$ such that $\Gamma_P(I) = I$. Additionally, the SMs also lack the properties of Cumulativity (the addition of lemmas from a model to the original program doesn't alter the semantics) and Relevancy (the possibility of implementing a top-down query-driven proof-procedure), and its computation, even brave-reasoning, is NP-Complete [19].

Three years passed since the introduction of the SMs until Alan van Gelder along with Kenneth Ross and John Schlipf proposed [40] the Well-Founded Semantics (WFS). The WFS overcomes all the problems described before and still maintains a strong relation with NMR formalisms. It is more expressive than the stable models semantics as it uses three-valued interpretations but is also a lot more skeptical, attributing the value of undefined to all those literals which are not undoubtfully true or false (or depend on ones who are, in turn, undefined), or lack classical support.

Several definitions of the well-founded semantics exist in the literature. For this work, we'll make use of the one defined by Baral and Subrahmanian [5], which is most closely related with our approach to the subject of this thesis.

Definition 14 (Baral-Subrahmanian operator Γ^2)**.** *Let P be a NLP and I an two-valued interpretation. The Baral-Subrahmanian operator* Γ^2 *is defined as being two applications of the Gelfond-Lifschitz*

operator Γ*:*

$$\Gamma_P^2(I) = \Gamma_P(\Gamma_P(I))$$

Definition 15 (Well-Founded Semantics)**.** *Let P be a NLP. The Well-Founded Semantics of program P is the tuple $\langle \mathcal{T}, \mathcal{TU} \rangle$, with $\mathcal{T}$ corresponding to the true literals according to the semantics and $\mathcal{TU}$ being the true or undefined literals according to the semantics.*

The $\mathcal{T}$ set can be calculated as being the least fixed point of Γ_P^2 starting with the empty interpretation, i.e., $\mathcal{T} = \Gamma_P^{2\uparrow\omega}(\{\})$, with ω being the least limit ordinal.

The $\mathcal{TU}$ set can be calculated as being the next iteration of Γ_P, starting from the $\mathcal{T}$ set, i.e., $\mathcal{TU} = \Gamma_P(\mathcal{T})$.

We will refer to the well-founded model M of any NLP P as $WFM_P(M) = \langle \mathcal{T}, \mathcal{TU} \rangle$. The $\mathcal{T}$ set of the WFM will be referred to as $\mathcal{T}_{WFM}$ and the set $\mathcal{TU}$ of the WFM will be referred to as $\mathcal{TU}_{WFM}$.

The well-founded semantics has all the properties the stable models miss, namely cumulativity, relevancy and existence. Furthermore, it ensures uniqueness of model, a property desirable for the well-founded semantics. Its well-founded model is computable in polynomial time and several WFS implementations ([35, 37, 39, 41]) have been studied and exist nowadays. Moreover, several important relations exist between the well-founded semantics and nonmonotonic reasoning formalisms [5, 12], as well as between the well-founded semantics and the stable models semantics [10,34]. Additionally, the WFS was later expanded to its explicit [2,28], paraconsistent [9] and disjunctive [1] counterparts. It falls out of the scope of this thesis to analyse and comment on these relations; the key subject here is that the WFS ended up being heavily studied.

During the last decades, these two semantics were extensively used and studied for nonmonotonic reasoning and knowledge representation. The stable models semantics on one side, when researchers needed a two-valued semantics, capable of representing several possible worlds as models of a program; and the well-founded semantics on the other side, when researchers were willing to sacrifice the expressiveness of having multiple two-valued models in order to have a three-valued skeptical and sure-footed semantics.

2.4 Enter the Revised Stable Models

The drawbacks of the stable models semantics, described in the previous section, are dealt with in the well-founded semantics but they aren't actually *solved*. Only in 2004 Pinto and Moniz Pereira proposed [30] a way of solving the problems of the stable models and have been studying [30–32] its aspects ever since.

The Revised Stable Models (rSMs) is a semantics for NLPs which applies the principle of *Reductio Ad Absurdum* (RAA) to solve the main problems of the stable models semantics, namely

inconsistency, cumulativity and relevancy. Inconsistency in SMs, as it was shown by François Fages [11], comes from the existence of odd loops over negation (OLONs) in the program such as

$$P = \{a \leftarrow\sim a\}$$

François Fages goes on explaining that not only OLONs are the source of the problem for inconsistency, but also infinite chains over negation (ICONs), as in the case of:

$$\begin{aligned} P = \{&p(X) \leftarrow p(s(X)) \\ &p(X) \leftarrow\sim p(s(X))\} \end{aligned}$$

In a nutshell, according to the principle of *reductio ad absurdum* if a certain hypothesis leads to a contradiction, it should be withdrawn. In the first case, by assuming $\sim a$ as true, this assumption will eventually lead us to conclude a, a contradiction. Therefore $\sim a$ cannot be concluded, so a should be. In the second case, if we assume $\sim p(X)$ then the bodies of both rules should be false, which is impossible since they are logical oppositions. Therefore, $p(X)$ must be true, as $\sim p(X)$ will never be.

Revised Stable Models are in everything similar to the stable models, only changing the way the semantics deals with OLONs and ICONs – it solves them using RAA reasoning, given that the conclusions derived from this resolution keep consistent with the rest of the program. This results in an extension to the stable models semantics, which solves all the problems detailed previously.

Definition 16 (Sustainable Set [31,32])**.** *As a shorthand notation only for this definition, let $WFM(P)$ denote the positive atoms of the Well-Founded Model of P, that is WFM(P) is the least fixed point of operator Γ_p^2 [5], i.e., Γ_p applied twice. Intuitively, we say a set S is sustainable in P if and only if any atom A in S does no go against the well-founded consequences of the remaining atoms in S whenever $S \setminus \{A\}$ itself is a sustainable set. The empty set by definition is sustainable. "Not going against" means that atom A cannot be false (A is true or undefined) in the well-founded model of $P \cup S \setminus \{A\}$, i.e., it belongs to set $\Gamma_{P \cup S \setminus \{A\}}(WFM(P \cup S \setminus \{A\}))$*

Formally we say S is sustainable if and only if:

$$\forall_{A \in S}, S \setminus \{A\} \textit{ is sustainable} \Rightarrow A \in \Gamma_{P \cup S \setminus \{A\}}\left(WFM\left(P \cup S \setminus \{A\}\right)\right)$$

If S is empty the condition is trivially true.

Definition 17 (Revised Stable Models and Semantics [31,32])**.** *Let $RAA_P(M) \equiv M \setminus \Gamma_P(M)$. M is a Revised Stable Model of an Normal Logic Program P, if and only if:*

- *M is a minimal classical model, with "$\sim$" interpreted as classical negation;*

- $\exists_{\alpha \geq 2}$ *such that* $\Gamma_P^{\alpha}(M) \supseteq RAA_P(M)$
- $RAA_P(M)$ *is sustainable.*

The Revised Stable Models semantics is the intersection of its models, just as in the stable models semantics.

We represent a revised stable model M of a NLP P as $rSM_P(M)$.

As it was proven in [32], the revised stable models enjoy not only the existence of model property, therefore loosing the inconsistency the SMs had, as well as the properties of cumulativity and relevancy. They also served as a preliminary proof that the principle of *reductio ad absurdum* could be used as an additional form of support, which is a very important result for the context of this thesis.

2.5 The Next Step

This semantics and the additional notion of support it employs, somewhat change the map of knowledge representation and nonmonotonic reasoning semantics we have detailed so far.

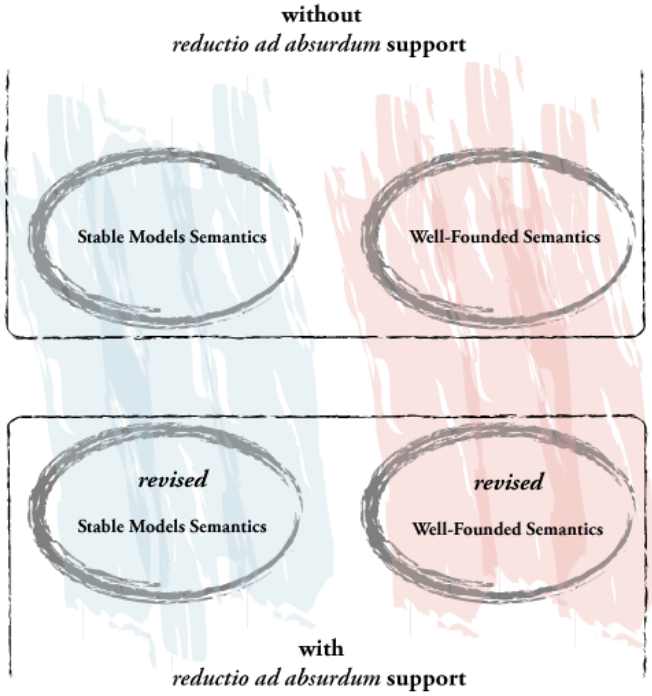

Figure 2.1: Map of transition and relations between the existing semantics for normal logic programs. Note that in the blue area, on the left, we have two-valued semantics while in the red area, on the right, we have the three-valued semantics. Additionally, the bottom of the figure represents semantics with RAA support, while the top of the figure represents those without RAA support.

The top level of figure 2.1 represents the current state of the art in semantics for normal logic programs *without* RAA support. The stable models represent a flexible two-valued semantics, capable of representing several possible worlds in its various models. The well-founded

semantics represents a single three-valued view of a normal logic program, enjoying several important properties but being too skeptical in its conclusions. The bottom level of the figure represents the corresponding semantics employing the principle of *reductio ad absurdum* as an additional form of support. The revised stable models solve all the problems the stable models had, becoming a semantics as flexible as the SMs, with some of the properties the WFS had. All it's missing to define is the Revised Well-Founded Semantics, a three-valued counterpart of the rSMs and the next transition point from the WFS, when using the RAA notion of support.

In the next chapter we'll discuss the motivation for the definition of a rWFS, and continue providing its declarative definition. We'll then discuss the properties it enjoys when compared with the WFS and present some examples of its calculation.

Chapter 3

Revised Well-Founded Semantics

Contents

Where we propose the Revised Well-Founded Semantics and explore its properties. We present a declarative definition, relate it with previous concepts and semantics, and show some basic examples of its calculation. Most of the results in this chapter are then further explored in chapter 4.

3.1 A Motivation for the Revised Well-Founded Semantics

As we've discussed in the previous chapter, we have three main semantics available for reasoning with normal logic programs, relevant for the context of this work. Two of these semantics (SMs and WFS) complement each other, by offering different levels of expressiveness and skepticism and by allowing an acceptable transition between two-valued and three-valued models. The other one (rSMs), extends the stable models semantics by introducing a new notion of support (*reductio ad absurdum*) in normal logic programs and thus solving the problems the SMs had. We now discuss the motivation for the introduction of a revised well-founded semantics, by identifying the main problems it should solve and the additional properties it will bring to the family of semantics with RAA support.

As we've seen, unlike the stable models semantics, the well-founded semantics is well defined for all normal logic programs. Every normal logic program has a well-founded model that specifies its meaning with three truth values where it states the truth, falsity and undefinedness of literals. Under the WFS, atoms are said to be true if and only if they are undoubtfully true, i.e., they have classical support and when transitioning to a two-valued context with several (possible) models, they are true in all models. That being so, literals are said to be undefined if they fall in one of the following cases:

- Either the literals are part of a two-valued choice (true in some worlds, false in others) but never undeniably true or false;
- Or the literals are not classically supported;
- Or the literals depend on an already undefined literal.

Otherwise, they are false.

It's easy to see that literals that are part of choices should be undefined – choices are made in a two-valued context so things are either true or false. In a three-valued context, as they can't be true and false at the same time they remain undefined. Additionally, if a literal depends on an already undefined literal it should also be undefined as undefinedness propagates throughout the program. However, lack of classical support could be overcome by the introduction of another form of support. This will be our first motivation.

Recalling the example programs in chapter 1 (page 1), by employing the additional form of support by absurdity (following the principle of *reductio ad absurdum*), programs with certain patterns[1] could have their truth value defined, thus resulting on more expressive results. Note that we are not claiming that the well-founded semantics is not expressive enough; on the contrary, we are claiming that if we enabled the WFS with an additional form of support, this would result on a more expressive semantics as it would be capable of giving the intended meaning to certain specific program patterns. By introducing the rWFS, we will have a way

[1] We will introduce these patterns formally in the next section.

of, in a three-valued scenario, transitioning from a semantics merely based on the classical notion of support to one with both the classical and the RAA notion of support. This is our first motivation, study the introduction of the RAA notion of support in the WFS.

Moreover, by employing this form of support, the revised well-founded semantics would also be capable of relating with the revised stable models semantics, and therefore defining a proper way of transitioning between a three-valued context to a two-valued context with RAA-support. We then argue that this is our second motivation for the introduction of the rWFS, the definition of a three-valued counterpart for the rSMs which, by making use of the *reductio ad absurdum* form of support, provides a more expressive result than the WFS on certain specific program patterns.

From these two motivations, we can see that the introduction of this semantics is heavily related with the notion of undefinedness. We argue that literals which should have the undefined value are those that fall in one of the following cases:

1. Either are true in some worlds and false in others, but never are undoubtfully true nor false;
2. Depend on an atom which already has an undefined truth value.

As a consequence, this results in a more expressive result for the rWFS when compared with the WFS: the universe of undefined literals, given a certain NLP, no longer includes those literals that can be solved by *reductio ad absurdum*.

Another consequence resulting from the introduction of the rWFS is the possibility of defining the Revised Partial Stable Models, in opposition to the Partial Stable Models. The Partial Stable Models (PSMs) represent the universe of choices which can be made in undefined literals when transitioning from the well-founded model to the stable models. If the notion of RAA is not present, some programs will have paths that will never reach a stable model, because of the presence of patterns which can only be solved by RAA. The introduction of a revised well-founded semantics will allow a correct transition from the single rWFM to the several rSMs, only by performing choices on the undefined literals. Note that this is precisely the point – undefined literals will now *only* represent choices that can be made and those choices will correspond to the several revised stable models.

Given this motivational scenario, the revised well-founded semantics will allow us to maintain two important parallels with the existing semantics:

1. It will correspond to the well-founded semantics with the *reductio ad absurdum* notion of support;
2. It will be a three-valued counterpart of the revised stable models.

Moreover, the revised well-founded semantics should be able to solve the following problems:

1. Be able to give a correct meaning to programs which require the *reductio ad absurdum* form of support;
2. Allow for a correct transition between a three-valued scenario to a two-valued one, where both scenarios share the *reductio ad absurdum* notion of support;
3. Allow the correct transition from a three-valued scenario without RAA support to one with RAA support.

At the same time, it will have the following additional properties:

1. Provide more expressive results than the well-founded semantics, due to its ability to revise the truth value of literals that were undefined but did not corresponded to choices in a two-valued context;
2. Allow for a complete definition of the revised partial stable models, where the path between the rWFM and the rSMs always exists and is only the result of choices on undefined literals.

Therefore, we propose that a revised well-founded semantics for normal logic programs should have, at least, the following properties:

- Be definable in a declarative way, by relating the principle of *reductio ad absurdum* with the well-founded semantics;
- Be also definable by a monotonic and continuous fixed point operator;
- Always be defined for any normal logic program (existence of the revised well-founded model), and be uniquely defined (uniqueness of the revised well-founded model);
- Comply with the properties of cumulativity and relevancy;
- Maintain the polynomial complexity of the well-founded semantics, regarding the calculation of the revised well-founded model;
- Coincide with the well-founded semantics for normal logic programs with no OLONs nor ICONs.

We believe this is the best starting point for studying the application of the RAA notion of support in a three valued scenario, as it allows us to start by comparing it with the existing semantics, study the relations existing between them and contextualize it within the existing work on semantics for normal logic programs.

We will continue this chapter by setting the principles and definitions this semantics is based on, most of them extended versions of the ones presented in [32], and then continue defining it in a declarative way. At the end of this definition we'll show how it keeps the properties we just set as essential and then show some calculation examples of this semantics. We finish with some conclusions and open issues.

3.2 Preliminary Definitions

We now proceed with the definition of several OLON and ICON patterns, which basically consist of their syntactic structure. These definitions are extended versions of the ones presented in [32].

3.2.1 Odd Loops Over Negation

Regarding OLONs, the simplest OLON possible is:

$$P = \{\lambda_0 \leftarrow\sim \lambda_0\}$$

This kind of OLON can be extended with an arbitrarily long chain of rules between λ_0 and $\sim\lambda_0$ like:

$$\begin{aligned} P = \{&\lambda_0 \leftarrow \lambda_1 \\ &\lambda_1 \leftarrow \lambda_2 \\ &\lambda_2 \leftarrow \ldots \\ &\ldots \leftarrow \lambda_n \\ &\lambda_n \leftarrow\sim \lambda_0\} \end{aligned}$$

We call the set of atoms $\lambda_i \ (1 \leq i \leq n)$ the RAA chain of support for the OLON where λ_0 is the head of the OLON. Additionally, every rule in the OLON may have a finite arbitrarily long conjunction of literals:

$$\begin{aligned} P = \{&\lambda_0 \leftarrow \lambda_1, PC_1(\lambda_0) \\ &\lambda_1 \leftarrow \lambda_2, PC_2(\lambda_0) \\ &\lambda_2 \leftarrow \ldots \\ &\ldots \leftarrow \lambda_n, PC_n(\lambda_0) \\ &\lambda_n \leftarrow\sim \lambda_0, PC_{n+1}(\lambda_0)\} \end{aligned}$$

where $PC_j(\lambda_0) \ (1 \leq j \leq n+1)$ is the conjunction of literals of the rule with head λ_{j-1} in the OLON. We call these rules the preconditions of the OLON and refer to it as $PC_j(\lambda_0)$ to mean the precondition of rule j for the OLON with the head λ_0. The head of the OLON is the literal which is actually provable by RAA, and we call if the core of this OLON. Indirect OLONs, as we'll see, may have several cores, one for each rSM. As long as all $PC_j(\lambda_0)$ are true, the OLON is in effect and λ_0 is true by RAA reasoning as it depends on its own negation. We call this kind of OLON a direct OLON since it has only one negative literal in its chain of support.

An indirect OLON is one where the number of negative literals present in the RAA chain of support is odd and greater than one. Essentially, just assume that some $\Lambda_i, (0 \leq i \leq n)$ literals are default negated λ_i literals, i.e., $\Lambda_i = \lambda_i \vee \Lambda_i = \sim \lambda_i$:

$$\begin{aligned} P = \{ & \lambda_0 \leftarrow \Lambda_1, PC_1(\lambda_0) \\ & \lambda_1 \leftarrow \Lambda_2, PC_2(\lambda_0) \\ & \lambda_2 \leftarrow \ldots \\ & \ldots \leftarrow \Lambda_n, PC_n(\lambda_0) \\ & \lambda_n \leftarrow \sim \lambda_0, PC_{n+1}(\lambda_0) \} \end{aligned}$$

Assuming all $PC_j(\lambda_0)$ are true, the meaning associated with this OLON corresponds to each $\lambda_i, (0 \leq i \leq n)$ being true (along with several consequences that follow from each λ_i). In order for each λ_i to be provable by RAA, all $PC_k(\lambda_0)$ must be true, with $k = \{i, i+2, i+4, \ldots\}$[2]. This kind of OLON has, at most, n revised stable models, all indirectly supported.

So, formally we have:

Definition 18 (Odd Loop Over Negation (OLON))**.** *An OLON is a set of rules in the form*

$$\begin{aligned} & \lambda_0 \leftarrow \Lambda_1, PC_1(\lambda_0) \\ & \lambda_1 \leftarrow \Lambda_2, PC_2(\lambda_0) \\ & \lambda_2 \leftarrow \ldots \\ & \ldots \leftarrow \Lambda_n, PC_n(\lambda_0) \\ & \lambda_n \leftarrow \Lambda_0, PC_{n+1}(\lambda_0) \end{aligned}$$

with $\Lambda_i, (0 \leq i \leq n)$ being either λ_i or $\sim \lambda_i$ and the number of negative literals in the RAA chain of support being odd. Additionally, it should hold that:

$$\begin{aligned} \forall_{r_1 \in OLON}, \exists_{r_2 \in OLON} : & \alpha \in head(r_1) \wedge \\ & (\alpha \in tail(r_2) \vee \sim \alpha \in tail(r_2)) \end{aligned}$$

We call the set $\{PC_1(\lambda_0), \ldots, PC_{n+1}(\lambda_0)\}$ the set of preconditions of the OLON and the set $\{\Lambda_0, \Lambda_1, \ldots, \Lambda_n\}$ the RAA chain of support for λ_0.

We will refer to all the preconditions of an OLON with head α as the set $PCs(\alpha)$.

Additionally, we say that an OLON is active or in effect, meaning that its head α can be concluded by reductio ad absurdum*, iff all the atoms in $PCs(\alpha)$ are true.*

We can also provide a definition for an indirect OLON and how we are going to represent it.

[2]Result from [32]

Definition 19 (Indirect OLON)**.** *An OLON O is an indirect OLON iff the number of negative literals of O is greater than one (written $NNL(O) > 1$). An indirect OLON can be represented in the form*

$$OLON^i(\{a_1, a_2, \ldots, a_n\})$$

with $a_1, a_2, \ldots, a_n$ being the several possible heads of the OLON. Note, therefore, that an indirect OLON has several possible heads.

Additionally, we can also provide the definition of a direct OLON, and how it can be represented.

Definition 20 (Direct OLON)**.** *An OLON O is a Direct OLON iff the number of negative literals of O is one (written $NNL(O) = 1$). A direct OLON can be represented in the form*

$$OLON^d(a)$$

with a being the head of the OLON.

From the previous definition results the definition of all the direct OLONs in a normal logic program.

Definition 21 (Set of All Direct OLONs of a NLP)**.** *Given a NLP P, the set OLs is the set of all Direct OLONs of P, written $OLONS_P^d = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, iff:*

$$OLONS_P^d = \{\alpha_i : OLON^d(\alpha_i)), (1 \leq i \leq n)$$

with each α_n being the head of each direct OLON.

3.2.2 Infinite Chains Over Negation

Another pattern that can be solved by RAA, and is also treated as undefined in the well-founded semantics, are infinite chains over negation (ICONs). The notion of ICON was first identified by François Fages in [11] as an example of a NLP which although not having any OLON still had no stable models:

$$\begin{aligned} P = \{ & p(X) \leftarrow p(s(X)) \\ & p(X) \leftarrow\sim p(s(X))\} \end{aligned}$$

The grounded version of this program is:

$$
\begin{aligned}
P = \{ & p(0) \leftarrow p(s(0)) \\
& p(0) \leftarrow\sim p(s(0)) \\
& p(s(0)) \leftarrow p(s(s(0))) \\
& p(s(0)) \leftarrow\sim p(s(s(0))) \\
& p(s(s(0))) \leftarrow p(s(s(s(0)))) \\
& p(s(s(0))) \leftarrow\sim p(s(s(s(0)))) \\
& \vdots \\
\}&
\end{aligned}
$$

The issue in this program is that each $p(X)$ is infinitely justified by each $p(s(X))$, and $p(s(X))$ by $p(s(s(X)))\ldots$

Reasoning by absurdity also takes place in ICONs. Let's consider any hypothesis $\sim p(X)$. For this hypothesis to be true, the tails of $p(X) \leftarrow \ldots$ would have to be false. In particular $\sim p(s(X))$ and $p(s(X))$ would have to be true. But this would turn $p(X)$ true, thus leading to a contradiction. So, by RAA, $p(X)$ must be true for all X.

This simple version of an ICON can be expanded in the following way:

$$
\begin{aligned}
P = \{ & p(X) \leftarrow \lambda_1, PC_{\lambda_1} && p(X) \leftarrow \mu_1, PC_{\mu_1} \\
& \lambda_1 \leftarrow \ldots && \mu_1 \leftarrow \ldots \\
& \ldots \leftarrow \lambda_n, PC_{\lambda_n} && \ldots \leftarrow \mu_m, PC_{\mu_m} \\
& \lambda_n \leftarrow p(s(X)), PC_{\lambda_{n+1}} && \mu_m \leftarrow\sim p(s(X)), PC_{\mu_{m+1}} \}
\end{aligned}
$$

where the meaning is always the same: $p(X)$ must be true, either by the chain of $\lambda_i, (1 \leq i \leq n)$ or by the one of $\mu_j (1 \leq j \leq m)$, provided that all the tails for the chain in question are true. As $p(X)$ ends up depending on mutually exclusive literals, $p(s(X))$ and $\sim p(s(X))$, either one of them has to be true, rendering $P(X)$ inevitably true.

Definition 22 (Infinite Chain Over Negation (ICON))**.** *An ICON is an infinite set of rules in the form*

$$
\begin{aligned}
& \alpha \leftarrow \lambda_1, PC_{\lambda_1} && \alpha \leftarrow \mu_1, PC_{\mu_1} \\
& \lambda_1 \leftarrow \ldots && \mu_1 \leftarrow \ldots \\
& \ldots \leftarrow \lambda_n, PC_{\lambda_n} && \ldots \leftarrow \mu_n, PC_{\mu_n} \\
& \lambda_n \leftarrow \beta, PC_{\lambda_{n+1}} && \mu_n \leftarrow\sim \beta, PC_{\mu_{n+1}}
\end{aligned}
$$

We call α *the* head of the ICON *and represent the ICON as* $ICON_P(\alpha)$*, where the meaning is:* α *is true by RAA if both of the sets* PCs_λ *or* PCs_μ *are true.*

Let's now define the set of all ICONs of a given NLP:

Definition 23 (Infinite Chains Over Negation of a NLP)**.** *Given a NLP P, ICs is the set of all literals of P which are heads of Infinite Chains Over Negation, written $ICONS_P = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ iff:*

$$ICONS_P = \{x : ICON_P(x)\}$$

Definition 24 (Preconditions function)**.** *Given any literal S where S is the head of an RAA pattern, $PC(S)$ is the set of all preconditions of S.*

3.3 Declarative Semantics

We proceed now with the declarative definition of the revised well-founded semantics. In this definition we recall the principle of *reductio ad absurdum*, which states that if by assuming some hypothesis as false that assumption leads us to a contradiction, then our hypothesis must be revised to true.

This is the principle used in the definition of the rSMs, the first of this *revised* family of semantics. The question now is if this principle is still valid in a three-valued context.

Following the same ideas behind the revised stable models semantics, the revised well-founded semantics intends to use RAA reasoning to revise literals that can't be false into true ones. However, now that we have three values of truth available to chose from, why should those literals be revised to true instead of undefined?

Let's start by looking at a simple case:

$$P = \{a \leftarrow \sim a\}$$

By reasoning by absurdity we say that a can't be false because that would lead us to a contradiction. We now have two truth values at our disposal, true and undefined. At this point, we argue that the undefined truth value should mean that there would be some worlds where a would be true and others where a would be false. However, we already seen that a cannot be false in any world, so it is not correct to opt for the undefined value. By absurdity, the only value we can chose from is true. Let's enunciate this principle:

Definition 25 (Reductio ad absurdum with three truth values)**.** *Consider a normal logic program P in a three-valued context. Additionally, consider that in this three-valued context the value of undefined is attributed to literals which depend on an already undefined literal or literals which are either true or false in a two-valued context.*

We say that any literal α is true by the principle of reductio ad absurdum *if we have to assume its falsity $\sim \alpha$ in order to conclude α.*

The reasoning behind this principle is the following: we start by considering the hypothesis $\sim \alpha$ and

assume that it allows us to conclude α, which is an absurd. We now say that α should be true instead of undefined because by undefined we would be considering the hypothesis that α might be false in some worlds, an hypothesis we already seen that is not possible.

This being the case, as α can't be false nor undefined it should be true by absurdity.

Let's now see what results we want to revise in our semantics. Direct OLONs only have a single rSM associated with them so, as they don't represent choices, they should be true in the revised well-founded model, provided that their preconditions are true.

Indirect OLONs have several rSMs, which represent the choices made in a two-valued context. The only way for an indirect OLON to always have a single model is if the other possible models have unsatisfied preconditions. This, however, breaks the OLON's chain of support, thus providing classical support for the indirect OLON. Therefore, indirect OLONs are not to be considered.

ICONS, as we've previously seen, have the singularity of ending in contradictory tails, thus resulting on a single conclusion from this pattern. Therefore, ICONs are also to be considered for the rWFM.

Given these principles, the only result we want to add to the WFS is the revision of undefined literals present in direct OLONs and ICONs, if they are in effect. An OLON or ICON is in effect if its preconditions set is acceptable, i.e., if each of these patterns depends on literals which either are already true in the model or, being undefined, will be revised to true after the application of the *reductio ad absurdum* principle.

We start by defining the set of RAA patterns which can be considered for the revised well-founded semantics:

Definition 26 (Considerable RAA Patterns set RAA_P)**.** *Let P be a normal logic program, $ICONS_P$ be the set of all infinite chains over negation of P and $OLONS_P^d$ be the set of all direct odd loops over negation of P.*

We define the set RAA_P of considerable RAA patterns of P as:

$$RAA_P = heads(ICONS_P) \cup heads(OLONS_P^d)$$

RAA_p is a set in the form $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, with $\alpha_i, 1 \leq i \leq n$ being the head of a given RAA pattern.

We now define the set of acceptable preconditions for RAA patterns. this i the set of preconditions we are, in principle, willing to accept as valid ones for RAA patterns:

Definition 27 (Acceptable preconditions for a set of RAA patterns)**.** *Let P be a normal logic program, RAA_p be the set of considerable RAA patterns of P and $WFM_P = \langle \mathcal{T}, \mathcal{T}\mathcal{U} \rangle$ be the well-founded model of P.*

We say that the literals which can be accepted as preconditions for RAA patterns form the set $accPC_P$, defined as:

$$accPC_P = \{\mathcal{T}_{WFM} \cup RAA_P\}$$

We now define the relation of conformity between literals and preconditions. This relation will allow us to determine which RAA patterns have valid preconditions.

Definition 28 (Conformity between literals and preconditions)**.** *Let $\check{=}$ mean the conformity between a literal and a set of preconditions S and, symmetrically, $\check{\neq}$ mean the conflict between a literal and a set of preconditions. We say, without loss of generality, that, given a certain literal X belonging to S:*

$$\begin{aligned} &X \check{=} X \\ &X \check{\neq} \{\} \\ \sim &X \check{=} \{\} \\ \sim &X \check{=} X \text{ iff } X \notin accRAA_P. \text{ Otherwise } \sim X \check{\neq} X \end{aligned}$$

Additionally, we say that all literals in a set $T = \{a_1, \ldots, a_n\}$, with $n \geq 2$ are loop conflicting *(and therefore conflicting) if:*

$$\begin{aligned} \forall_{a_i \in T}, \exists_{a_j \in T} : \sim a_j \in PC(a_i) \wedge \\ a_i \neq a_j \wedge \\ 1 \leq i \leq n \wedge \\ 1 \leq j \leq n \end{aligned}$$

Finally, the empty set is never conflicting with anything.

Definition 29 (Acceptable RAA Patterns set $accRAA_P$)**.** *Let P be a Normal Logic Program and RAA_P be the set of considerable RAA patterns of P.*

The acceptable RAA patterns of P is the subset $accRAA_P$ of RAA_P which is defined as:

$$\begin{aligned} \forall_{x \in RAA_P}, &x \in accRAA_P \text{ iff} \\ &\forall_{a \in PC(x), b \in accPC_P}, a \check{=} b \end{aligned}$$

What we've defined so far is a way of pruning the set of RAA patterns whose undefinedness we are willing to revise. Basically we accept RAA patterns whose preconditions are either true or undefined being that, if they are undefined, they should clearly be part of the acceptable RAA patterns set. A certain pattern is acceptable if its preconditions are in conformity with the acceptable preconditions set.

In order to better understand these definitions, let's consider the following example:

$$P = \{a \leftarrow\sim a$$
$$b \leftarrow\sim a, \sim b\}$$

and the sets generated from P

$$RAA_P = \{a, b\}$$
$$accPC_P = \{\} \cup \{a, b\} = \{a, b\}$$
$$PC(a) = \{\}$$
$$PC(b) = \{\sim a\}$$

We now intend to generate the set $accRAA_P$ with patterns whose preconditions are not conflicting. Let's see in detail what we mean by *conflicting*: the first OLON has no preconditions, so it can be added directly to $accRAA_P$, because $\{\}$ is in conformity with everything. The second OLON has a precondition of $\sim a$, which conflicts with one of the literals in $accPC_P$. Therefore it will not be part of $accRAA_P$.

Let's consider another example:

$$P = \{a \leftarrow\sim a, z$$
$$b \leftarrow\sim a, \sim b\}$$

and the sets generated from P

$$RAA_P = \{a, b\}$$
$$accPC_P = \{\} \cup \{a, b\} = \{a, b\}$$
$$PC(a) = \{z\}$$
$$PC(b) = \{\sim a\}$$

The second OLON has a precondition of $\sim a$. By the fourth rule of definition 28, $\sim a \check{=} a$ iff $a \notin accRAA_P$. Then let's see if a belongs to $accRAA_P$. The first OLON has a precondition of z. As $z \notin accPC_P$, we have that $z \check{\neq} \{\}$, and therefore $a \notin accPC_P$. Therefore, $b \in accRAA_P$.

This demonstrated the generation of the $accRAA_P$ set. We now define the revised well-founded

semantics for normal logic programs:

Definition 30 (Revised Well-Founded Model and Semantics)**.** *Let P be a normal logic program, WFM_P be the well-founded model of P and $accRAA_P$ be the set of acceptable RAA patterns of P.*

The revised well-founded model of P is the the tuple $\langle \mathcal{T}^r, \mathcal{TU}^r \rangle$ where $\mathcal{T}^r$ corresponds to the true literals under the revised well-founded semantics and $\mathcal{TU}^r$ corresponds to to true or undefined literals under the revised well-founded semantics. The revised well-founded model of P is the well-founded model of the NLP $P \cup (accRAA_P \cap \mathcal{TU}_{WFM})$. Formally we have:

$$rWFM_P = WFM_{P \cup (accRAA_P \cap \mathcal{TU}_{WFM})}$$

Under the revised well-founded semantics, an atom is true if it belongs to the set $\mathcal{T}_{rWFM}$, is undefined if it belongs to the set $\mathcal{TU}_{rWFM} \setminus \mathcal{T}_{rWFM}$ and is false if it doesn't belong to $\mathcal{TU}_{rWFM}$.

Let's examine this definition in more detail. The intuition we are following is that if certain literals obey certain conditions they should be added to the program as facts and the program's WFM should calculated so that the interference of these facts may be considered.

The first part of the set of facts we're adding, $accRAA_P$, derives from the previous definitions. Knowing that we already have a set of acceptable RAA patterns, whose preconditions are either true or some considerable RAA pattern, we are willing to add them to the program as these are the kind of patterns we are revising.

However, we intersect this set with the true or undefined set that comes from the WFM – $(accRAA_P \cap \mathcal{TU}_{WFM})$. The reason for this is that even if a certain literal is part of a direct OLON or an ICON and its preconditions are acceptable, if by some other reason the OLON is false in the well-founded model it should never be revised to true. Recalling the idea behind this thesis, what is intended with this semantics is precisely the revision of undefined literals, those that while not proven to be undoubtfully false, can't be proven to be true by the classical notion of support. So all the literals we're interested in are in the $\mathcal{TU}_{WFM}$ set. By intersecting the set $accRAA_P$ with $\mathcal{TU}_{WFM}$ we make sure that the only atoms we're adding are those that, more than just being provable by RAA reasoning, are not undoubtfully false.

Finally, we join this set of facts with the original program – $P \cup (accRAA_P \cap \mathcal{TU}_{WFM})$. This is done because of the extensive nature of the rWFS, i.e., it extends the WFS for NLPs with OLONs or ICONs but it provides the same results as the WFS in a NLP without such patterns. In other words, all that was true in WFS will never cease to be true. What can happen is the revision of certain undefined literals to true and others to false because of the additional form of support now available. Moreover, the atoms that come from the set $accRAA_P \cap \mathcal{TU}_{WFM}$ will never contradict the atoms in the set $\mathcal{T}_{WFM}$ as they already were true or undefined under the WFS. This means that, either they were already true, and their value remains, or they were undefined, and in that case all relevant atoms regarding those were also undefined.

This ends our declarative definition of the revised well-founded semantics. We now present

the properties this semantics enjoys and proceed with a set of illustrative examples.

3.4 Properties

We now demonstrate in detail the proofs for the main properties we claim the rWFS enjoys.

3.4.1 Well-Founded Semantics Extension for Programs Without RAA Patterns

Given a Normal Logic Program P without any of the RAA patterns identified in section 3.2, M the well-founded model of P and M^r the revised well-founded model of P, we will prove that $rWFM_P(M^r) \Leftrightarrow WFM_P(M)$:

Theorem 1 (Revised Well-Founded Semantics extends Well-Founded Semantics for programs without RAA Patterns)**.** *Consider $M = \langle \mathcal{T}, \mathcal{TU} \rangle$ the well-founded model of a NLP P, denoted by WFM_P. Additionally consider $M^r = \langle \mathcal{T}^r, \mathcal{TU}^r \rangle$ the revised well-founded model of P, denoted by $rWFM_P$ where the sets of considerable RAA patterns is empty: $RAA_P = \emptyset$.*

It holds that $M = M^r$.

Proof. Considering the declarative definition of the rWFS we have that $M^r = WFM_{P \cup (accRAA_P \cap \mathcal{TU}_{WFM})}$, being that $accRAA_P$ is a subset of RAA_P. However, we also know that $RAA_P = \emptyset$ and therefore $accRAA_P = \emptyset$. Then,

$$
\begin{aligned}
& WFM_P(M) && = rWFM_P(M^r) \Leftrightarrow && (3.1)\\
\Leftrightarrow\ & WFM_P(M) && = WFM_{P \cup (accRAA_P \cap \mathcal{TU}_{WFM})}(M^r) \Leftrightarrow && (3.2)\\
\Leftrightarrow\ & WFM_P(M) && = WFM_{P \cup (\emptyset \cap \mathcal{TU}_{WFM})}(M^r) \Leftrightarrow && (3.3)\\
\Leftrightarrow\ & WFM_P(M) && = WFM_{P \cup \emptyset}(M^r) \Leftrightarrow && (3.4)\\
\Leftrightarrow\ & WFM_P(M) && = WFM_P(M^r) && (3.5)\\
\Leftrightarrow\ & M && = M^r && (3.6)
\end{aligned}
$$

□

3.4.2 Existence and Uniqueness of Model

The existence and uniqueness of model are two notions that derive directly from the well-founded semantics, i.e., these are two properties that the WFS enjoys and that it's intended for the rWFS to enjoy as well.

Theorem 2 (The Revised Well-Founded Model always exists and is unique)**.** *Consider a normal logic program P, where $WFM_P(M)$ denotes the program's well-founded model and $rWFM_P(M^r)$ denotes the program's revised well-founded model.*

M^r always exists and is unique.

Proof. Theorem 1 proved that if P has no RAA Patterns, its rWFM corresponds to its WFM. This being the case, it is trivially proven that its rWFM always exists and is unique because being its WFM, it already enjoys the properties of existence and uniqueness.

Furthermore, if P has any of the RAA Patterns we are considering for the rWFS, we already now that its rWFM is the WFM of a transformed program with some literals added as facts. This being the case, the rWFS is again given by the WFS, a semantics that already enjoys the properties of existence and uniqueness of model. □

3.4.3 Cumulativity

Consider $rWFM_P(M^r) = \langle \mathcal{T}^r, \mathcal{T}\mathcal{U}^r \rangle$ the revised well-founded model of a normal logic program P. The property of cumulativity states that if any atom a belongs to $\mathcal{T}^r$, it can be added to the original program P as a fact and the semantics of P will remain the same. Formally

$$\forall_{a,b \in \mathcal{T}^r} : rWFM_{P \cup \{a\}} = \langle \mathcal{T}^{r'}, \mathcal{T}\mathcal{U}^{r'} \rangle \wedge \\ b \in \mathcal{T}^{r'}$$

Theorem 3 (The Revised Well-Founded Semantics is cumulative)**.** *The revised well-founded semantics is a cumulative semantics.*

Proof. We start with the case where P has no RAA patterns whatsoever. We already seen that in this case $rWFS_P(M^r) = WFS_P(M)$, so the property is trivially satisfied because the Well-Founded Semantics already is cumulative.

This not being the case, we now that $rWFM_P(M^r) = WFM_{P \cup (accRAA_P \cap \mathcal{T}\mathcal{U}_{WFM})}(M^r)$, meaning that the revised well-founded semantics will be the result of the well-founded semantics of a program to which we added a set of facts. This being the case, the property is again trivially satisfied as the WFS is cumulative. □

3.4.4 Relevancy

The property of relevancy states that the semantics of a NLP P with respect to a set of literals L only depends on the set of relevant rules of P regarding L.

We start by defining the set of relevant rules of a NLP regarding a literal[3].

[3]Definition borrowed from [32].

Definition 31 (Set of Relevant rules Rel(P,L))**.** *Given a normal logic program P and a set of atoms L, the set of relevant rules $Rel(P,L)$ is defined as:*

$$\begin{aligned} Rel(P,L) = \{R_i \in P : & head(R_i) = a \wedge a \in L\} \cup \\ \{Rel(P,X) : & R_i \in P \wedge \\ & head(R_i) = a \wedge a \in L \wedge \\ & ((X \in body(R_i)) \vee (\sim X \in body(R_i)))\} \end{aligned}$$

Formally, and for the case of the revised well-founded semantics:

$$\begin{aligned} \forall_{L \subseteq M^r} : & rWFS_P(M^r) \Rightarrow \\ & SEM_L^{rWFS}(P) = SEM_L^{rWFS}(Rel(P,L)) \end{aligned}$$

Theorem 4 (The Revised Well-Founded Semantics is relevant)**.** *Let $SEM_L^{rWFS}(P)$ mean the semantics of all literals in set L according to the rWFS*[4] *and let $Rel(P,L)$ mean the set of rules relevant to a certain set L of literals in P. Finally, it holds that the Well-Founded Semantics is relevant, i.e.,*

$$\begin{aligned} \forall_{L \subseteq M} : & WFS_P(M) \Rightarrow \\ & SEM_L^{WFS}(P) = SEM_L^{WFS}(Rel(P,L)) \end{aligned}$$

The revised well-founded semantics is a relevant semantics.

Proof. With the results we have so far (theorems 1, 2 and 3) we can say that if P has no RAA Pattern, by theorem 1 the revised well-founded semantics is relevant.

This not being the case, we have the set $accRAA_P$ non-empty. However, as the rWFS corresponds to the WFS of a transformed program $P \cup (accRAA_P \cap \mathcal{TU}_{WFM})$, it will keep the property of relevancy, derived from the WFS. □

3.5 Examples

We now analyse some examples showing the results when applying the intuitions set forth in the declarative definition.

Example 1 (OLON with inner OLON[5])**.** *In this example we have a direct OLON over a with one of*

[4] Recal from the definition of the Revised Well-Founded Semantics, in page 29.
[5] This example is due to Sérgio Lopes

the literals in its chain of support also being an OLON (x).

$$P = \{a \leftarrow x \qquad WFM_P(M) = \langle\{\},\{a,x,y\}\rangle$$
$$x \leftarrow y, \sim x \qquad RAA_P = \{a,x\}$$
$$y \leftarrow \sim a\} \qquad accRAA_P = \{a\}$$

$$P^r = \{a \leftarrow x \qquad rWFM_P(M^r) = \langle\{a\},\{a\}\rangle$$
$$x \leftarrow y, \sim x$$
$$y \leftarrow \sim a$$
$$a \leftarrow\}$$

It's easy to see that the precondition for x is not valid as it doesn't belong to $\mathcal{T}_{WFM} \cup RAA_P$. Therefore, a is the only acceptable RAA pattern. As it intersects with $\mathcal{T}\,\mathcal{U}_{WFM}$ it is added to the original program.

Example 2 (Normal OLON)**.** *In this example we have an OLON over a and several other literals dependending both positively and negatively on a. According to the well-founded semantics, everything is undefined but, because the revised well-founded semantics employs RAA reasoning, in the revised well-founded model all literals will end up defined.*

$$P = \{a \leftarrow x \qquad WFM_P(M) = \langle\{\},\{a,x,b,c,d\}\rangle$$
$$x \leftarrow \sim a \qquad RAA_P = \{a\}$$
$$b \leftarrow \sim a \qquad accRAA_P = \{a\}$$
$$c \leftarrow \sim b$$
$$d \leftarrow \sim c\}$$

$$P^r = \{a \leftarrow x \qquad rWFM_P(M^r) = \langle\{a,c\},\{\}\rangle$$
$$x \leftarrow \sim a$$
$$b \leftarrow \sim a$$
$$c \leftarrow \sim b$$
$$d \leftarrow \sim c$$
$$a \leftarrow\}$$

Example 3 (OLON with an undefined dependency)**.** *In this program we have an odd loop over a which depends positively on the odd loop over b. This example shows that the revised well-founded*

semantics can correctly deal with this dependency although the dependency of b is an undefined one.

$$P = \{a \leftarrow b, x \qquad WFM_P(M) = \langle\{\},\{a,b,x\}\rangle$$
$$x \leftarrow\sim a \qquad RAA_P = \{a,b\}$$
$$b \leftarrow\sim b\} \qquad accRAA_P = \{a,b\}$$

$$P^r = \{a \leftarrow b, x \qquad rWFM_P(M^r) = \langle\{a,b\},\{\}\rangle$$
$$x \leftarrow\sim a$$
$$b \leftarrow\sim b$$
$$a \leftarrow$$
$$b \leftarrow\}$$

Example 4 (Indirect Odd Loop Over Negation)**.** *This example shows an indirect OLON. As indirect OLONs are never considered for the set RAA_P, no modification is ever done to the program so $rWFM_P(M^r) = WFM_P(M)$.*

$$P = \{a \leftarrow\sim b \qquad WFM_P(M) = \langle\{\},\{a,b,c\}\rangle$$
$$b \leftarrow\sim c \qquad RAA_P = \{\}$$
$$c \leftarrow\sim a\} \qquad accRAA_P = \{\}$$

$$P^r = P \qquad rWFM_P(M^r) = WFM_P(M) = \langle\{\},\{a,b,c\}\rangle$$

Example 5 (Direct OLON with illegal precondition)**.** *This program shows an OLON which will never be part of the model because it depends negatively on another OLON. Note that all OLONs were undefined and in the rWFS, as one of them is revised to true, the other is revised to false.*

$$P = \{a \leftarrow\sim a \qquad WFM_P(M) = \langle\{\},\{a,b\}\rangle$$
$$b \leftarrow\sim b, \sim a\} \qquad RAA_P = \{a,b\}$$
$$accRAA_P = \{a\}$$

$$P^r = \{a \leftarrow\sim a \qquad rWFM_P(M^r) = \langle\{a\},\{a\}\rangle$$
$$b \leftarrow\sim b, \sim a$$
$$a \leftarrow\}$$

Example 6 (An OLON with another OLON inside[6])**.** *In this example we have two OLONs, being that the one of x is inside the one of a. a will end up being part of the rWFM, not by RAA but by classical*

[6]This example is due to Sérgio Lopes

reasoning, although it is part of the acceptable OLONs.

$$
\begin{aligned}
P = \{ & a \leftarrow x \\
& x \leftarrow y \\
& y \leftarrow \sim a \\
& y \leftarrow \sim x \}
\end{aligned}
\qquad
\begin{aligned}
& WFM_P(M) = \langle \{\}, \{a,x,y\} \rangle \\
& RAA_P = \{a,x\} \\
& accRAA_P = \{a,x\}
\end{aligned}
$$

$$
\begin{aligned}
P^r = \{ & a \leftarrow x \\
& x \leftarrow y \\
& y \leftarrow \sim a \\
& y \leftarrow \sim x \\
& a \leftarrow \\
& x \leftarrow \}
\end{aligned}
\qquad
rWFM_P(M^r) = \langle \{a,x\}, \{a,x\} \rangle
$$

3.6 Concluding remarks

In this chapter we proposed and defined the revised well-founded semantics. We showed that this definition extends the well-founded semantics in its properties and results, and corresponds to the ideas we set in the beginning of this chapter. The definition of what is a revised well-founded model recurs only to the ideas of OLONs and ICONs which, although being easy to reason about at a conceptual level, are not trivial to identify in larger, more complicated programs. Because of this, and the desire to further relate the rWFS with the revised stable models, we present a transformational semantics for the rWFS in the next chapter, as well as a set of additional definitions and properties.

[This page was intentionally left blank]

Chapter 4

Transformational semantics and additional properties

Contents

Where we propose a syntactic transformation based on a new property of the revised stable models, which enables us to calculate the revised stable models and the revised well-founded model of a program. While providing this bridge to the previous results of the rSMs, we also show that all their properties still maintain, as well as those of the rWFS.

4.1 Introduction

In the previous chapter we defined the Revised Well-Founded Semantics, a three-valued extension of the Revised Stable Models Semantics. This definition, although being easy to reason about at a conceptual higher level, is not easy to translate in terms of a set of automated operations. This difficulty results on the unavoidable operational notion behind RAA patterns - the notion of *odd loop* and *chain of support*. These notions are not easy to deal with by means of automatic operations because they involve going through chains of support to detect RAA patterns. For arbitrarily long programs, the complexity cost of these operations would be immense – only to see if literals are in OLONs, not to mention the actual calculation of the model.

However, at the core of the definition stands the calculation of the well-founded model of the original program united with certain RAA patterns. This suggests that if the problem of detecting RAA patterns could be solved, a transformational semantics would be easier to define.

In this chapter we study such transformational semantics. We start with a fresh look on RAA patterns and study their behavior on the revised stable models semantics. We then discuss the property of *RAA Rule Extension*, unknown in the context of the rSMs and use this property to define a syntactic theoretical approach to solving RAA patterns. With these results, we define an extension of the Γ operator– the Γ^r operator, and show how it can be used to calculate rSMs and the rWFM of a NLP. With these two new definitions we show that the properties of rSMs and the rWFS still hold and show examples of its calculation. We then conclude with some open issues.

4.2 Further Explorations in Revised Stable Models

From the RAA patterns we have identified so far ($OLONs^d$, $OLONs^i$ and $ICONs$), we use $OLONs^d$ and $ICONs$ in the calculation of the rWFM. The solving of each of these patterns by RAA is actually the calculation of the revised stable model of each pattern, which we add to the original program when calculating the rWFS. This is why we don't consider $OLONs^i$ – they generally result in several different models which we don't want to consider as part of the revised well-founded model[1].

One of the ways of looking at the problem of why programs with RAA patterns don't have stable models, is because RAA patterns can't be solved with classical support. Therefore, one possible approach to this problem is trying to determine what a program would need to have so that RAA patterns would have classical support. This is the motivation behind the property of RAA Rule Extension. This property applies to all program patterns solvable by *reductio ad absurdum* and consists on complementing each pattern with a single rule whose meaning is the reasoning behind its RAA resolution. This rule, we'll later see in this section, if added to the original program will allow the stable models of the transformed program to coincide with the

[1]Several rSMs represent choices at a two-valued level, i.e., undefinedness at a three-valued level.

revised stable models of the original program. This result is of great importance since it not only means that RAA reasoning for a given program can be captured with a single rule, but also that given this transformation, the stable models semantics will always coincide with the revised stable models thus enjoying all the properties it missed.

4.2.1 RAA Rule Extension

We'll now introduce the property of RAA Rule Extension through several examples of RAA pattern situations.

Example 7 (Simple Direct OLON)**.** *Let's recall the RAA patterns we described in section 3.2. The reasoning behind OLONS, whether they are direct or indirect, is always the same: if a certain atom is dependent on its own negation it should be true provided that the set of preconditions (possibly) present in the OLON are also true.*

To this idea we now add the following: the head of the OLON will be true if, in particular, the tail of the OLON (the head of the rule that depends on the negation of the head of the OLON) is false. As an example consider:

$$\begin{aligned} P = \{ & a \leftarrow x \\ & x \leftarrow y \\ & y \leftarrow\sim a\} \end{aligned}$$

We know that a is true by RAA reasoning. It fails as a stable model because it lacks classical support. Now consider adding the following rule to the program:

$$a \leftarrow\sim y$$

This rule states that a should be true by reductio ad absurdum *if y is false. In other words, having identified an OLON in a, we know that its RAA chain of support is to be false. In particular, the last literal in the RAA chain of support has to be false, thus rendering the whole chain false. Because y is true if a is false, we end up generating an even loop over negation between y and a. It will never actually become an even loop because y will imply a set of atoms that eventually results in a, thus being contradictory. Therefore, y will never be true and a will be. Our revised program would therefore be:*

$$\begin{aligned} P^r = \{ & a \leftarrow x \\ & x \leftarrow y \\ & y \leftarrow\sim a \\ & a \leftarrow\sim y\} \end{aligned}$$

What this rule added to the program was the reasoning behind the RAA resolution of literal a. This resolution has to go through two steps: the first one being the identification of the OLON,

and the second one the identification of the conditions that have to be satisfied in order for a certain literal that belongs to an OLON to have classical support. a, which is part of an OLON, is true if y is false. We call y the tail of the OLON.

Example 8 (Direct OLON with preconditions)**.** *If we have preconditions in our OLON, they have to be true in order for the OLON to be active. In these cases, the rule we add must contemplate this. Consider a new program:*

$$\begin{aligned} P = \{ & a \leftarrow x, i \\ & x \leftarrow y, j \\ & y \leftarrow\sim a, k\} \end{aligned}$$

Besides the fact that y has to be false, it's also necessary for i, j and k to be true. Therefore, the rule we generate is:

$$a \leftarrow\sim y, i, j, k$$

where $\sim y$ is the tail of the RAA chain of support and i, j and k are the preconditions of the OLON.

Example 9 (Preconditions which are RAA chains of support)**.** *When contemplating preconditions, we have the special case when preconditions are chains of support for the same OLON:*

$$\begin{aligned} P = \{ & a \leftarrow x, y \\ & x \leftarrow z \\ & y \leftarrow w \\ & z \leftarrow\sim a \\ & w \leftarrow\sim a\} \end{aligned}$$

Apparently, following the reasoning we set in the previous examples, we should add the rule

$$a \leftarrow\sim z, y, \sim w, x$$

However, both x and y are preconditions that result in the same OLON, so each of them is already implied by the fact that it belongs to the OLON's chain of support. So, the rule we have to add is

$$a \leftarrow\sim z, \sim w$$

generating the program:

$$\begin{aligned} P = \{ & a \leftarrow x, y \\ & x \leftarrow z \\ & y \leftarrow w \\ & z \leftarrow\sim a \\ & w \leftarrow\sim a \\ & a \leftarrow\sim z, \sim w\} \end{aligned}$$

Example 10. *Consider another example:*

$$\begin{aligned} P = \{ & a \leftarrow x \\ & x \leftarrow y, z \\ & y \leftarrow\sim a \\ & z \leftarrow\sim z\} \end{aligned}$$

In this case, z is part of an OLON chain of support but it is not for the same head. Therefore, we add it as a precondition for the OLON of a. As z is also an OLON, but a direct one without any other rules in between, we simply add it as a fact, because there is no tail of the RAA chain of support here.

$$\begin{aligned} & a \leftarrow\sim y, z \\ & z \leftarrow \end{aligned}$$

Example 11 (Indirect OLONs)**.** *So far we only dealt with direct OLONs. How are indirect OLONs dealt with?*

$$\begin{aligned} P = \{ & a \leftarrow\sim b \\ & b \leftarrow\sim c \\ & c \leftarrow\sim a\} \end{aligned}$$

P has three revised stable models, which have its core in the three literals that can be proven by RAA. The intuition so far is the same as in the previous examples: a is provable by RAA if c is false:

$$a \leftarrow\sim c$$

b is true if a is false:

$$b \leftarrow\sim a$$

and finally, c is true if b is false:

$$c \leftarrow\sim b$$

So these are the three rules we need to add to P.

Example 12 (Indirect OLON with preconditions)**.** *What happens if we have preconditions? Consider:*

$$\begin{aligned} P = \{&a \leftarrow\sim b \\ &b \leftarrow\sim c, x \\ &c \leftarrow\sim a\} \end{aligned}$$

Apparently x should be added to all rules as part of the preconditions set. However, x only interferes with rSMs with b and c cores[2]*: assuming $\sim a$ as true gives us c. Independently of the truth value of x, b will always be false thus rendering a true by RAA. The rules to add are:*

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

The idea is that as we go on through the RAA chain of support we add the preconditions of each rule every two rules. Consider another example of an indirect OLON with preconditions:

$$\begin{aligned} P = \{&a \leftarrow\sim b \\ &b \leftarrow\sim c, x \\ &c \leftarrow\sim a \\ &x \leftarrow\sim x, a\} \end{aligned}$$

The rules to add are:

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

We don't add x to the rule for b because it eventually derives in $\sim b$, another path to the same RAA chain.

Example 13 (ICONs)**.** *Considering now ICONs, we know that the reasoning is somewhat different, as*

[2]Result from [32], also mentioned in section 3.2 and definition 18

the rules we add are not dependent on the head of the ICON but on the mutually exclusive tails. Given:

$$\begin{aligned} P = \{&\alpha \leftarrow \lambda_1, PC_{\lambda_1} && \alpha \leftarrow \mu_1, PC_{\mu_1} \\ &\lambda_1 \leftarrow \ldots && \mu_1 \leftarrow \ldots \\ &\ldots \leftarrow \lambda_n, PC_{\lambda_n} && \ldots \leftarrow \mu_n, PC_{\mu_n} \\ &\lambda_n \leftarrow \beta, PC_{\lambda_{n+1}} && \mu_n \leftarrow \sim \beta, PC_{\mu_{n+1}}\} \end{aligned}$$

If we have identified an ICON over α, the rules to add are simply:

$$\begin{aligned} \lambda_n &\leftarrow PC_{\lambda_{n+1}}, PC_{\mu_{n+1}} \\ \mu_n &\leftarrow PC_{\mu_{n+1}}, PC_{\lambda_{n+1}} \end{aligned}$$

which will render α true because the contradictions it had are now solved. In fact, these rules are just modified versions of the last rules in the ICON, the ones with contradictory tails. Note that we demand that both tails are true in any ICON, because the ICON is only in effect if both tails are in effect[3]*. In the case of ICONs, we say that the tail of an ICON is formed by the two rules which have contradictory bodies in the ICON.*

Let's now define the function $Tail^{OLON}(X)$ which will give the Tail of an OLON:

Definition 32 (Tail function for OLONs)**.** *Let P be a NLP and X be the head of an OLON (direct or indirect) of P.*

We define the function $Tail^{OLON}(X)$ the following way:

$$\begin{aligned} Tail^{OLON}(X) = \{h : &h = head(r) \wedge \\ &\sim X \in body(r)\} \end{aligned}$$

Now we define the function $Tail^{ICON}(X)$ which will give the Tail of an ICON:

Definition 33 (Tail function for ICONs)**.** *Let P be a NLP and Y the head of an ICON of P.*

We define $Tail^{ICON}(Y)$ as:

$$\begin{aligned} Tail^{ICON}(Y) = \{\lambda_n, \mu_n : &\lambda_n = head(r_1) \wedge \\ &\mu_n = head(r_2) \wedge \\ &r_1 \text{ and } r_2 \text{ have contradictory tails.}\} \end{aligned}$$

We refer to the first tail of an ICON as $Tail^{ICON}_{\lambda}(Y)$ and the second tail of an ICON as $Tail^{ICON}_{\mu}(Y)$.

[3] Result from [32] and definition 22 on page 24.

Given these three RAA patterns, $OLONS^d$, $OLONS^i$ and $ICONS$, we can now define a theoretical operator which adds the substitution rules to the original program, for each RAA pattern found:

Definition 34 (Substitution Operator Σ)**.** *Given a NLP P and the sets of RAA patterns $OLONS^d$, $OLONS^i$ and $ICONS$, the $\Sigma(P)$ operator adds the following substitution rules to P:*

For each $OLON^d(a)$ add the following rule:

$$a \leftarrow\sim Tail^{OLON}(a), PCs_a$$

For each head of $OLON^i(\{a_1, a_2, \ldots, a_n\})$ add the following rule:

$$a_i \leftarrow\sim Tail^{OLON}(a_i), PC_{a_i}$$

For each $ICON_P(\alpha)$ add the following rules a copy of the rules with contradictory tails, where the contradictory literals are removed and the preconditions of both rules are present in both bodies of both rules:

$$Tail^{ICON}_{\lambda}(\alpha) \leftarrow PC_{\lambda_{n+1}}, PC_{\mu_{n+1}}$$
$$Tail^{ICON}_{\mu}(\alpha) \leftarrow PC_{\mu_{n+1}}, PC_{\lambda_{n+1}}$$

where $Tail^{ICON}_{\lambda}$ and $Tail^{ICON}_{\mu}$ are the tails of the ICON of α.

We call P^r the transformed program by the Σ operator and define $\Sigma(P) = P^r$.

We called this a theoretical operator because there is still the problem associated with finding the RAA patterns. The idea with this operator is to define a framework form which we'll define a program transformation to find RAA patterns.

We now present a new definition for the revised stable models, based on this operator:

Definition 35 (Alternative definition of the Revised Stable Models and Semantics)**.** *Given a NLP P, M is a revised stable model of P iff M is a stable model of $\Sigma(P)$, i.e., $\Gamma_{\Sigma(P)}(M) = M$.*

Theorem 5 (Soundness and completeness regarding the revised stable models semantics)**.** *According to the rule extension property, programs with RAA patterns have all their patterns extended with a rule that provides them with classical support. Then, on this revised version of the original program, we calculate the stable models.*

The stable models of P^r are the revised stable models of P.

Proof. We start with recalling the definition of the revised stable models, which states that M is a rSM of a normal logic program P if

- M is a minimal classical model, with "$\sim$" interpreted as classical negation;
- $\exists_{\alpha \geq 2} : \Gamma_P^{\alpha}(M) \supseteq RAA_P(M)$;
- $RAA_P(M)$ is sustainable.

with $RAA_P(M) = M \setminus \Gamma_P(M)$.

From this definition results that $M = \Gamma_P(M) \cup RAA_P(M)$ is a revised stable model if:

- M is a minimal classical model, with "$\sim$" interpreted as classical negation;
- $\exists_{\alpha \geq 2} : \Gamma_P^{\alpha}(M) \supseteq RAA_P(M)$;
- $RAA_P(M)$ is sustainable.

If these three conditions hold then M is a revised stable model.

If M is a revised stable model, then it enjoys the property of cumulativity, from which results that:

$$\begin{aligned} M = \Gamma_P(M) \cup RAA_P(M) = \\ = \Gamma_{P \cup RAA_P(M)}(M) \end{aligned}$$

This result allows us to present an alternative view of the definition of the revised stable models:

Let $RAA_P(M)$ be the set of atoms in M which don't have classical support in P, i.e., $RAA_P(M) = M \setminus \Gamma_P(M)$.

$RAA_P(M)$ is a set which contains literals present in RAA patterns and literals which depend on those but lack classical support because of their dependency on RAA patterns. Then, we can say that the set $RAA_P(M)$ is made up of the subsets $RAA_P^{class}(M)$ (literals lacking classical support but not present in RAA patterns) and $RAA_P^{raa}(M)$ (literals which lack classical support but are present in RAA patterns):

$$RAA_P(M) = RAA_P^{class}(M) \cup RAA_P^{raa}(M)$$

The atoms in $RAA_P^{class}(M)$ will be true in the revised stable model because of their dependency on atoms in $RAA_P^{raa}(M)$, which allows us to say that, in fact, only the set $RAA_P^{raa}(M)$ needs to be added to P in order for the rSM to be calculated.

$M = \Gamma_{P \cup RAA_P^{raa}(M)}(M)$ is a revised stable model of P iff:

- M is a minimal classical model, with "$\sim$" interpreted as classical negation;
- $\exists_{\alpha \geq 2} : \Gamma_P^{\alpha}(M) \supseteq RAA_P(M)$;
- $RAA_P(M)$ is sustainable.

Knowing this, we'll now prove that, for a NLP P, and a revised stable model M^r of P,

$$rSM_P(M^r) \Leftrightarrow SM_{\Sigma(P)}(M)$$

Consider $rSM_P(M^r)$. Then, $M^r = \Gamma_{P \cup RAA_P^{raa}(M^r)}(M^r)$.

On the other hand, we have that $M = \Gamma_{\Sigma(P)}(M)$. $\Sigma(P)$ is a program transformation that adds to P a set of rules which may conclude heads of RAA patterns. Let's call this set of rules R_p. We then have

$$\Gamma_{\Sigma(P)}(M) = \Gamma_{P \cup R_P}(M)$$

At this point, proving that $rSM_P(M^r) \Leftrightarrow SM_{\Sigma(P)}(M)$ is equivalent to proving that $(P \cup RAA_P^{raa}(M^r)) \Leftrightarrow (P \cup R_P)$.

R_P is a set of rules which allows us to conclude atoms present in RAA patterns. These rules have the forms:

$$\begin{aligned}
&a \leftarrow\sim Tail^{OLON}(a), PCs_a, \text{ for direct OLONs} \\
&a_i \leftarrow\sim Tail^{OLON}(a_i), PC_{a_i}, \text{ for indirect OLONs and} \\
&Tail_{\lambda}^{ICON}(\alpha) \leftarrow PC_{\lambda_{n+1}}, PC_{\mu_{n+1}} \text{ and} \\
&Tail_{\mu}^{ICON}(\alpha) \leftarrow PC_{\mu_{n+1}}, PC_{\lambda_{n+1}} \text{ for ICONs}
\end{aligned}$$

The set of heads of rules of R_P is a superset of $RAA_P^{raa}(M^r)$ which, in turn, does not include any literal that is part of a RAA pattern but which won't be part on a revised stable model. Te reason for these literals not being true in the rSM has to do with these RAA patterns not being active. In particular, an RAA pattern is not active if its preconditions don't hold. They fail to be in the $RAA_P^{raa}(M^r)$ set because if their preconditions don't hold and they are part of $RAA_P^{raa}(M^r)$, then $RAA_P^{raa}(M^r)$ won't be sustainable. Consider as an example:

$$P = \{a \leftarrow\sim b, \sim a$$
$$b \leftarrow\sim b\}$$

and consider that $M^r = \{a,b\}$. then, $RAA_P(M^r) = \{a,b\}$. However, $\{a,b\}$ is not sustainable as, according to the definition of sustainability[4], $a \notin \Gamma_{P\cup RAA_P(M^r)\setminus\{a\}}(WFM(P\cup RAA_P(M^r)\setminus\{a\}))$, because the OLON of a demands $\sim b$ as a precondition.

So, while $RAA_P^{raa}(M^r)$ only has atoms whose preconditions were satisfied, R_P has rules for all atoms but these rules include the demand that the preconditions be satisfied. If they are not, these atoms will never be concludable. This is because of the definition of sustainability, which we rewrite here in a somewhat simpler form.

A set S is sustainable iff:

$$\forall_{\alpha\in S}, \alpha \in \Gamma_P(WFM_{P'}) \wedge P' = P\cup S\setminus\{\alpha\}$$

From this definition follows that, if $\alpha \in S$ and $PC(\alpha)$ doesn't hold, then S is not sustainable. If $PC(\alpha)$ doesn't hold, α won't be true. In a program where the fact α is not added, $\alpha \notin \Gamma_P(WFM_{P'})$.

Additionally, it also holds that if S is not sustainable, then $\exists_{\alpha\in S} : PC(\alpha)$ doesn't hold. If S is not sustainable, then $\exists_{\alpha\in S} : \alpha \notin \Gamma_P(WFM_{P'})$. In order for $\alpha \notin \Gamma_P(WFM_{P'})$, i.e., not being present in a stable model, then either all rules with head α were cut from P' or there exist rules with head α such that their bodies don't hold. In either case, their preconditions don't hold.

Given this result, we conclude that

$$P\cup RAA_P^{raa}(M^r) \Leftrightarrow P\cup R_P$$

and therefore, the stable models of $P\cup RAA_P^{raa}(M^r)$ are the same as the stable models of $P\cup R_P$. In particular, given M^r a revised stable model of P, if $M^r = \Gamma_{P\cup RAA_P^{raa}(M^r)}(M^r)$ then $M^r = \Gamma_{P\cup R_P}(M^r)$.

This result allows us to conclude the other side of the implication, that if M is a stable model of $P\cup R_P$, then M is a revised stable model of P because, in particular, M is also a stable model of $P\cup RAA_P^{raa}$, and therefore is a revised stable model.

□

[4]Definition 16 on page 14.

Now that RAA patterns are natural stable models of transformed programs, we also define the revised well-founded semantics based on this program transformation. Our approach is the same used in [5] which defines the Well-Founded Semantics as being the least fixed point of Baral-Subrahmanian operator Γ^2. We now apply the same ideas for calculating the rWFS:

Definition 36 (Alternative definition of the Revised Well-Founded Model and Semantics)**.** *Given a NLP P, the revised well-founded model of P is the tuple $\langle \mathcal{T}, \mathcal{TU} \rangle$, with $\mathcal{T}$ (true literals) being the first fixed point of operator $\Gamma^2_{\Sigma(P)}$ starting from the empty interpretation, and $\mathcal{TU}$ (true or undefined literals) being the next iteration of $\Gamma_{\Sigma(P)}$ from the first fixed point of $\Gamma^{r^2}_{\Sigma(P)}$, i.e., $\Gamma_{\Sigma(P)}(\mathcal{T})$.*

Theorem 6 (Soundness and completeness regarding the revised well-founded semantics)**.** *Given a normal logic program P, it holds that its rWFM can be calculated by the Γ^2 operator applied on $\Sigma(P)$:*

$$rWFM_P = \langle \Gamma^{2\uparrow\omega}_{\Sigma(P)}(\{\}), \Gamma_{\Sigma(P)}(\Gamma^{2\uparrow\omega}_{\Sigma(P)}(\{\})) \rangle$$

Proof. We know, by definition 30 on page 29, that the rWFM of any NLP P is given by the WFM of P extended with all $OLONS^d$ and $ICONS$ with acceptable preconditions.

Let's start by considering that P has no RAA patterns whatsoever. In that case:

$$rWFM_P(M^r) = WFM_P(M)$$

and

$$WFM_P(M) = \langle \Gamma^{2\uparrow\omega}_{P}(\{\}), \Gamma_{P}(\Gamma^{2\uparrow\omega}_{P}(\{\})) \rangle$$

However, if no RAA patterns exist, $P = \Sigma(P)$ which allows us to conclude that:

$$rWFM_P(M^r) = \langle \Gamma^{2\uparrow\omega}_{\Sigma(P)}(\{\}), \Gamma_{\Sigma(P)}(\Gamma^{2\uparrow\omega}_{\Sigma(P)}(\{\})) \rangle$$

Let's now consider that P has RAA patterns. As the rWFM is the WFM of an extended program $P \cup (accRAA_P \cap \mathcal{TU}_{WFM})$ we have:

$$rWFM_P(M^r) = \langle \Gamma^{2\uparrow\omega}_{\Sigma(P)}(\{\}), \Gamma_{\Sigma(P)}(\Gamma^{2\uparrow\omega}_{\Sigma(P)}(\{\})) \rangle$$

which is equivalent to saying that

$$\langle \Gamma^{2\uparrow\omega}_{accRAA_P \cap \mathcal{T}\mathcal{U}_{WFM}}(\{\}), \Gamma_{accRAA_P \cap \mathcal{T}\mathcal{U}_{WFM}}(\Gamma^{2\uparrow\omega}_{accRAA_P \cap \mathcal{T}\mathcal{U}_{WFM}}(\{\}))\rangle = \langle \Gamma^{2\uparrow\omega}_{\Sigma(P)}(\{\}), \Gamma_{\Sigma(P)}(\Gamma^{2\uparrow\omega}_{\Sigma(P)}(\{\}))\rangle \quad (4.1)$$

Which, in turn, is true if

$$\Gamma^{2\uparrow\omega}_{accRAA_P \cap \mathcal{T}\mathcal{U}_{WFM}}(\{\}) = \Gamma^{2\uparrow\omega}_{\Sigma(P)}(\{\})$$

and

$$\Gamma_{accRAA_P \cap \mathcal{T}\mathcal{U}_{WFM}}(\Gamma^{2\uparrow\omega}_{accRAA_P \cap \mathcal{T}\mathcal{U}_{WFM}}(\{\})) = \Gamma_{\Sigma(P)}(\Gamma^{2\uparrow\omega}_{\Sigma(P)}(\{\}))$$

From $\Sigma(P)$ we know that it is a program transformation that adds to P a set of rules which may conclude heads of RAA patterns. Let's call this set of rules R_p. We then have

$$\Sigma(P) = P \cup R_P$$

On the other hand, we have that $accRAA_P \cap \mathcal{T}\mathcal{U}_{WFM}$ is a subset of R_P, since it includes all the $OLONS^d$ and $ICONS$ which have valid preconditions.

Then, expression 4.1 will be true if, in particular, only those rules with heads in the set $accRAA_P \cap \mathcal{T}\mathcal{U}_{WFM}$ will succeed.

R_P is a set of rules which allows us to conclude atoms present in RAA patterns. These rules have the forms:

$a \leftarrow\sim Tail^{OLON}(a), PCs_a$, for direct OLONs
$a_i \leftarrow\sim Tail^{OLON}(a_i), PC_{a_i}$, for indirect OLONs and
$Tail^{ICON}_{\lambda}(\alpha) \leftarrow PC_{\lambda_{n+1}}, PC_{\mu_{n+1}}$ and
$Tail^{ICON}_{\mu}(\alpha) \leftarrow PC_{\mu_{n+1}}, PC_{\lambda_{n+1}}$ for ICONs

For any generic rule $\alpha \leftarrow body$ in a normal logic program P, α will be part of the fixed point of Γ^2 starting from the empty interpretation if it belongs to the well-founded model of P.

In particular for $OLONS^i$, as they have a structure like

$$a \leftarrow\sim b$$
$$b \leftarrow\sim c$$
$$c \leftarrow \ldots$$
$$\ldots \leftarrow\sim a$$

each iteration of Γ^2 will cut all the rules in the indirect OLON[5]:

$$\Gamma_P^2(\{\}) = \Gamma_P(\Gamma_P(\{\})) = \Gamma_P(\{a,b,c,\ldots\}) = \{\}$$

We are then left with direct OLONs and ICONs. In the rWFS definition, the set of direct OLONs and ICONs is pruned to include only those patterns whose preconditions are not conflicting and which intersect with the $\mathcal{TU}_{WFM}$ set. However, the rules we added impose that the preconditions of each pattern are true, in order for it to become true:

$$a \leftarrow\sim Tail^{OLON}(a), PCs_a$$
$$Tail_{\lambda}^{ICON}(\alpha) \leftarrow PC_{\lambda_{n+1}}, PC_{\mu_{n+1}}$$
$$Tail_{\mu}^{ICON}(\alpha) \leftarrow PC_{\mu_{n+1}}, PC_{\lambda_{n+1}}$$

It's easy to see, for starters, that any $\alpha \notin \mathcal{TU}_{WFM}$ will never be concludable. If $\alpha \notin \mathcal{TU}_{WFM}$ and was on a RAA pattern, then one of its preconditions was false in WFM_P. In that case none of

$$a \leftarrow\sim Tail^{OLON}(a), PCs_a$$
$$Tail_{\lambda}^{ICON}(\alpha) \leftarrow PC_{\lambda_{n+1}}, PC_{\mu_{n+1}}$$
$$Tail_{\mu}^{ICON}(\alpha) \leftarrow PC_{\mu_{n+1}}, PC_{\lambda_{n+1}}$$

will be true.

Additionally, if any of the preconditions in PC(x) are not in conformity with another RAA pattern, then this means that:

- Either $PC(x)$ imposes some α and α is false;
- Or $PC(x)$ imposes some $\sim\alpha$ and α is a true RAA pattern.

[5]This behavior is similar to that of Γ^2 on ELONs.

In either case, $PC(x)$ will be false, thus rendering α false.

This proves that from R_P, only the rules with the heads in $accRAA_P \cap \mathcal{TU}_{WFM}$ will succeed as the least fixed points of Γ^2 which, in turn, proves that that the revised well-founded model can be calculated as the well-founded model of $\Sigma(P)$.

The converse implication is established in an analogous way.

□

The definition of this theoretical operator skipped over an important step, that of actually discovring RAA patterns in a program, in order to introduce an important property: that if we ad a rule to a progran with RAA patterns which has the RAA reasoning implicit in it, the all the SMs of this extended program are the rSMs of the original program.

We skipped that important step in order to deal with it now in a syntactic manner. In he next section we'll introduce the notion of counterfactual programs to allow to overcome the problem of identifying RAA patterns and generate syntactic transformations that are equivalent to the previous ones.

4.3 Transformational Semantics and Additional Properties

As we've seen in the previous section, although the property of rule extension allows for a complete transformational definition of the revised stable models and the revised well-founded semantics, there is still the problem of finding out whether or not a certain set of rules corresponds to an OLON or ICON. In this section we'll overcome this problem by introducing a solution based on counterfactual programs. We start by presenting the motivation and general working of counterfactual programs and then expose the transformation for each pattern we identified previously. We then redefine the revised stable models and the revised well-founded semantics based on this transformation.

4.3.1 Counterfactual Programs

Our approach when defining a fixed point operator for the rSMs and the rWFS is to express the principle of RAA in terms of a syntactic operation. This syntactic operation must provide a way for a certain literal X to be concludable if the assumption $\sim X$ was made, thus allowing us to overcome the problem of finding whether or not a certain literal can be proven by *reductio ad absurdum*. We'll accomplish this by means of counterfactual programs.

Counterfactual programs (CFPs) are copies of a NLP P, where we assume some default literal $\sim X$ true, i.e., we remove this default assumption from all rules of the counterfactual program. Roughly speaking, if X is now concludable then X was provable by RAA. These programs

are going to be used modularly, i.e., different CFPs are used to find different RAA patterns and they will provide a layer of reasoning which only allows the corresponding X to be concludable. This is, in fact, the idea behind the principle of *reductio ad absurdum*.

Example 14 (A simple counterfactual program)**.** *Consider any normal logic program P with any RAA pattern, for example:*

$$\begin{aligned} P = \{&\lambda_0 \leftarrow \lambda_1 \\ &\lambda_1 \leftarrow \lambda_2 \\ &\lambda_2 \leftarrow \ldots \\ &\ldots \leftarrow \lambda_n \\ &\lambda_n \leftarrow\sim \lambda_0\} \end{aligned}$$

Its counterfactual version is the following program, from which we removed the counterfactual hypothesis:

$$\begin{aligned} Pcf(\lambda_0) = \{&\lambda_0 \leftarrow \lambda_1 \\ &\lambda_1 \leftarrow \lambda_2 \\ &\lambda_2 \leftarrow \ldots \\ &\ldots \leftarrow \lambda_n \\ &\lambda_n \leftarrow\} \end{aligned}$$

Given this transformation, λ_0 now has classical support. In the following subsections we'll introduce the complete transformation for each RAA pattern we have studied so far.

Following the same reasoning behind the property of rule extension of the rSMs, we are going to define a *revised* version of the Γ operator (Γ^r operator) which performs this program transformation before we calculate the stable models.

Transforming Direct OLONs

Following the idea behind counterfactual programs, given a certain program P_1, let's calculate its counterfactual program:

$$\begin{aligned} P = \{&a \leftarrow x \\ &x \leftarrow y, \sim x \\ &y \leftarrow\sim a \\ &b \leftarrow c, \sim b \\ &c \leftarrow\sim c\} \end{aligned}$$

Let's start by analyzing this example. The first three rules are the program presented in example 1 in page 32. In this example we have a direct OLON over a with one of the literals in its chain of support also being an OLON (x). It's easy to see that the precondition for x is not valid and therefore, a is the only acceptable RAA pattern in the first three rules. As it intersects with $\mathcal{TU}_{WFM}$ it is added to the original program. The last two rules are a direct OLON with head b which has a precondition of c, and another direct OLON with head c. Both b and c will be true in the rWFM, as well as a, i.e., $rWFM_P(M^r) = \langle\{a,b,c\},\{a,b,c\}\rangle$. Additionally, P also has only one revised stable model: $rSM_P(M^r) = \{a,b,c\}$.

What the RAA principle states is that if we assume $\sim a$ and this leads us to a, then a must be true. However, in this case, we have several literals which have the potential to be provable by RAA, i.e., they appear at least once in the head of a rule and at least once negated in the tail of a rule. Therefore, we'll create counterfactual programs for all of them. As we don't want the literals in our counterfactual program to interfere with the literals in the original program, we are going to change them into an extended version – every literal X will now have a "d" in its superscript to indicate they are part of a *direct* OLON counterfactual program, as well as a natural number in its subscript to indicate which literal is this counterfactual program referring to. Note that we create counterfactual programs for each literal which we suspect that can be provable by *reductio ad absurdum*.

$$
\begin{aligned}
Pcf(a^d) = \{ & a_1^d \leftarrow x_1^d \\
& x_1^d \leftarrow y_1^d, \sim x \\
& y_1^d \leftarrow \\
& b_1^d \leftarrow c_1^d, \sim b \\
& c_1^d \leftarrow \sim c\}
\end{aligned}
\qquad
\begin{aligned}
Pcf(x^d) = \{ & a_2^d \leftarrow x_2^d \\
& x_2^d \leftarrow y_2^d \\
& y_2^d \leftarrow \sim a \\
& b_2^d \leftarrow c_2^d, \sim b \\
& c_2^d \leftarrow \sim c\}
\end{aligned}
$$

$$
\begin{aligned}
Pcf(b^d) = \{ & a_3^d \leftarrow x_3^d \\
& x_3^d \leftarrow y_3^d, \sim x \\
& y_3^d \leftarrow \sim a \\
& b_3^d \leftarrow c_3^d \\
& c_3^d \leftarrow \sim c\}
\end{aligned}
\qquad
\begin{aligned}
Pcf(c^d) = \{ & a_4^d \leftarrow x_4^d \\
& x_4^d \leftarrow y_4^d, \sim x \\
& y_4^d \leftarrow \sim a \\
& b_4^d \leftarrow c_4^d, \sim b \\
& c_4^d \leftarrow \}
\end{aligned}
$$

It's easy to see now that, for example, a_1^d can only be concluded given the precondition of $\sim x$ and the original assumption of $\sim a$, which is correct according to the revised stable model of this program ($rSM_P(M^r) = \{a,b,c\}$).

Note, that we kept the default literals unchanged because for all that matters in finding direct OLONs, default negations are not part of the RAA chain of support[6] – there is only one negation

[6] Recall definition 20 on page 23.

and that one is assumed true in the CFP. So the other negations are kept unchanged to allow the CFP to work as much as possible the same way as the original one – the only difference is that we are not interested in concluding any of those extended literals. In fact, we're only interested in knowing if, for example, a is part of a direct OLON. We can go one step further and say that, according to the property of RAA rule extension, a is provable by RAA if it is in an OLON, if its preconditions are satisfied and if the tail of the OLON is false. The imposition of the preconditions being satisfied is implicit in the counterfactual program as it would be impossible for a_1^d to be true if its preconditions wouldn't have been satisfied. So, the only things that have to happen for a certain literal in a direct OLON to be provable by RAA is to impose that the tail of the OLON must be false and the head of the counterfactual program must be true. Our new version of the extension rules presented in definition 34 is the following set of rules:

$$
\begin{aligned}
aOLON &\leftarrow a_1^d, \sim y \\
xOLON &\leftarrow x_2^d \\
bOLON &\leftarrow b_3^d \\
cOLON &\leftarrow c_4^d
\end{aligned}
$$

where the head of each rule refers to a certain literal being in an OLON and the body of the rule corresponds to the reasoning in definition 34, i.e., something is in an OLON (and is provable by RAA) if it is concludable by a counterfactual program and if the tail of the OLON is false. Note that these are simplified versions of the extension rules from which we removed the preconditions and to which we added the head of each counterfactual program.

However, one thing is still missing. This set of rules, as it is, will never allow us to conclude b_3^d because the rule $c_3^d \leftarrow\sim c$ will be removed by Γ operator (in turn, because c is part of the interpretation). However, c_4^d is directly concluded meaning that c is provable by RAA. So, what is missing is a way of propagating the conclusions we already reached on other CFPs. We'll add these propagation rules rules to each literal in a CFP which also has a CFP of its own, except for the literal of the CFP we're adding rules to (otherwise we would generate rules such as $X_\iota^d \leftarrow X_\iota^d$):

$$\text{Propagation rules for a} = \{x_1^d \leftarrow x_2^d \quad b_1^d \leftarrow b_3^d \quad c_1^d \leftarrow c_4^d\}$$

$$\text{Propagation rules for x} = \{a_2^d \leftarrow a_1^d \quad b_2^d \leftarrow b_3^d \quad c_2^d \leftarrow c_4^d\}$$

$$\text{Propagation rules for b} = \{a_3^d \leftarrow a_1^d \quad x_3^d \leftarrow x_2^d \quad c_3^d \leftarrow c_4^d\}$$

$$\text{Propagation rules for c} = \{a_4^d \leftarrow a_1^d \quad x_4^d \leftarrow x_2^d \quad b_4^d \leftarrow b_3^d\}$$

With the propagation of RAA results, we can now say what is true by *reductio ad absurdum*:

$$aRAA \leftarrow aOLON$$
$$xRAA \leftarrow xOLON$$
$$bRAA \leftarrow bOLON$$
$$cRAA \leftarrow cOLON$$

$$a \leftarrow aRAA$$
$$x \leftarrow xRAA$$
$$b \leftarrow bRAA$$
$$c \leftarrow cRAA$$

Let's now formalize this transformation and its intuitions. We start with what we mena by the peotential for a literal to be part of an OLON:

Definition 37 (Potential for RAA provability because of OLON belonging)**.** *Let P be a normal logic program with l rules in the form $r_k = H \leftarrow A_1, \ldots, A_m, \sim B_1, \ldots, \sim B_n$ ($m, n \geq 0$ and $1 \leq k \leq l$) and x an atom.*

We say that x has the potential for being proven by reductio ad absurdum *because of being part of an odd loop over negation iff:*

$$\exists_{r_1, r_2 \in P} : \sim X \in body(r_1) \wedge$$
$$X = head(r_2)$$

Where r_1 and r_2 may be the same rule.

Let's now specify how a counterfactual program for direct OLONs should be created.

Definition 38 (Direct counterfactual program Pcf)**.** *Let P be a normal logic program, X an atom in P which can be proven by* reductio ad absurdum *and ι (iota) an index identifying uniquelly X in the set of literals of P that can be proven by RAA.*

The counterfactual program for direct OLONs of X is constructed from a copy $Pcf(X^d)$ of P by performing the following operations on $Pcf(X^d)$:

1. *Remove from all rules of $Pcf(X^d)$ the default literal $\sim X$;*
2. *Replace all atoms A in all rules with the auxiliary atom A^d_ι;*

We call X^d_ι the head of this counterfactual program. The heads of the rules from which $\sim X$ was removed form the set $Tails_P(X)$.

As we are interested in generating counterfactual programs for the entire program and for all literals in the program whih have potential for being part of a direct OLON, we define the set of direct counterfactual programs.

Definition 39 (Direct counterfactual programs Set)**.** *Let P be a normal logic program, S the set of literals in P which have the potential to be proven by* reductio ad absurdum *and an ordering which associates each literal in S with a number $\iota \geq 0$.*

The set of all direct counterfactual programs of P regarding all literals in S is the set

$$\beta cf^d_P = \left(\bigcup_\iota Pcf(X^d_\iota) \right)$$

with ι being the program's counterfactual index, incremented as each new literal is considered.

We now define the propagation rules which are added to each counterfactual program.

Definition 40 (Propagation rules transformation for βcf^d_P)**.** *Let βcf^d_P be the set of all direct counterfactual programs of the NLP P.*

The propagation rules transformation for each counterfactual program in βcf^d_P consists of the following operation:

- *For each extended literal x^d_ι such that there exists a counterfactual program $Pcf(x^d)$ with head x^d_κ add the rule $x^d_\iota \leftarrow x^d_\kappa$ to the counterfactual program being considered.*

Finally, the rule extension transformation ,which corresponds to the generation of an equivalent rule to those of the Rule Extension property of the revised stable models.

Definition 41 (Rule extension transformation for βcf_P^d)**.** *Let βcf_P^d be the set of all direct counterfactual programs of the NLP P.*

The rule extension transformation for each counterfactual head X_ι^d in βcf_P^d consists of generating the following rule:

$$XOLON \leftarrow X_\iota^d, \sim Tails(X)$$

where $\sim Tails(X)$ corresponds to $\sim t_1, \sim t_2, \ldots, \sim t_n$ if $Tails(X) = t_1, t_2, \ldots, t_n$.

Additionally, for each XOLON generated, the following rules are also to be created:

$$XRAA \leftarrow XOLON$$
$$X \leftarrow XRAA$$

With these definitions we have a way of generating a program transformation capable of identifying direct OLONs:

Definition 42 (Direct OLONs transformation via counterfactual programs)**.** *Let P be a normal logic program.*

The Direct OLONs transformation via counterfactual programs of P is the revised program P^d which can be generated by the following operations:

1. *Create the counterfactual program set βcf_P^d;*
2. *Apply the propagation rules transformation and Rule extension transformation on βcf_P^d;*
3. *Let $P^d = \beta cf_P^d$.*

We say that the direct OLONs transformation via counterfactual programs for program P is the extended program P^d.

Theorem 7 (Soundness and completeness of the Direct OLONs transformation)**.** *From definition 34 on page 44 resulted that for each direct OLON found, a rule in the form $a \leftarrow\sim Tail^{OLON}(a), PCs_a$ was added to the program.*

The program transformation defined in definition 41 instead adds a rule stating $XOLON \leftarrow\sim Tails(X), X_\iota^d$.

It holds that the set of rules added by the direct OLONs transformation via counterfactual programs is equivalent to those of the RAA rule extension property for direct OLONs.

Proof. The rules added by the transformation operate on two levels. The first level is that of the counterfactual transformation in itself, and the second is from the extension rules onward.

On the counterfactual level we have, for a given NLP P, several copies of P where each literal which has the potential for being provable by RAA is assumed as false.

It is first important to show that this transformation won't allow atoms to be true if they are not provable by RAA. Consider, for example, that a literal x is true in the Pcf but wasn't in P. Then it became true by assuming its falsity, which means it was provable by *reductio ad absurdum*. Otherwise, assume that x is part of a RAA pattern but is not true by the counterfactual program. Then, x had no support under the counterfactual program, which means that either $PC(x)$ was false or x was not in a RAA pattern.

This point made, all X_ι^d which are true had to be part of $OLONs^d$ and had to had their preconditions true.

This means that in the second level only are true those literals which were part of direct OLONs and had their preconditions true.

Then, the conditions implied by

$$a \leftarrow\sim Tail^{OLON}(a), PCs_a, \text{ given that a is the head of a direct OLON}$$

are satisfied by

$$XOLON \leftarrow\sim Tails(X), X_\iota^d$$
$$XRAA \leftarrow XOLON$$
$$X \leftarrow XRAA$$

□

Transforming Indirect OLONs

By the previous transformation we were able to transform a program so that literals that had the potential to be proven by RAA and were in direct OLONs could be concluded, i.e., were given classical support. We'll discuss now a similar transformation for indirect OLONs.

Indirect OLONs, because of having a number of negative literals in their RAA chain of support greater than one, can't be concluded with the previous transformation, essentially because of

the way it deals with default negated literals. Consider, for example, the following program and its counterfactual versions for a, b and c:

$$\begin{aligned} P = \{ & a \leftarrow\sim b \\ & b \leftarrow\sim c \\ & c \leftarrow\sim a\} \end{aligned}$$

$$\begin{aligned} Pcf(a^d) = \{ & a_1^d \leftarrow\sim b \\ & b_1^d \leftarrow\sim c \\ & c_1^d \leftarrow\} \end{aligned} \qquad \begin{aligned} Pcf(b^d) = \{ & a_2^d \leftarrow \\ & b_2^d \leftarrow\sim c \\ & c_2^d \leftarrow\sim a\} \end{aligned} \qquad \begin{aligned} Pcf(c^d) = \{ & a_3^d \leftarrow\sim b \\ & b_3^d \leftarrow \\ & c_3^d \leftarrow\sim a\} \end{aligned}$$

For the known rSMs $\{a,b\}$, $\{b,c\}$ and $\{c,a\}$, the several rules in each counterfactual program would be removed by Γ operator, unabling any of the interpretations to become models. This happens because indirect OLONs have their support over more than one negation, which although being considered in order to prove that a certain literal is in an indirect OLON, it is not relevant for the conclusions. Therefore, the negation in the indirect OLONs also has to be auxiliary:

$$\begin{aligned} Pcf(a^i) = \{ & a_1^i \leftarrow\sim b_1^i \\ & b_1^i \leftarrow\sim c_1^i \\ & c_1^i \leftarrow\} \end{aligned} \qquad \begin{aligned} Pcf(b^i) = \{ & a_2^i \leftarrow \\ & b_2^i \leftarrow\sim c_2^i \\ & c_2^i \leftarrow\sim a_2^i\} \end{aligned} \qquad \begin{aligned} Pcf(c^i) = \{ & a_3^i \leftarrow\sim b_3^i \\ & b_3^i \leftarrow \\ & c_3^i \leftarrow\sim a_3^i\} \end{aligned}$$

This way, the test of whether a certain literal is part of an indirect OLON or not is local and independent of the global interpretation we are testing.

On indirect OLONs we are also interested in propagating the OLONs we already proven, whether these are direct or indirect. This supports the existence of programs with direct and indirect OLONs depending on each other. However the rules we had for direct OLONs are not enough for this propagation. In some cases we might want to allow for a direct resolution of OLONs in order to conclude some indirect OLON if (and only if) this resolution is not global, i.e., the literal we are solving is not part of our interpretation. As this resolution is a direct one, we use the direct version of our counterfactual transformation, however imposing on each rule that the literal it is refering to cannot be concluded. This results in another set of rules to be added to our indirect counterfactual programs:

$$\begin{aligned}\text{Local direct resolution rules for a} = \{&a_1^i \leftarrow\sim b, \sim a\\ &b_1^i \leftarrow\sim c, \sim b\\ &c_1^i \leftarrow\sim c\}\end{aligned}$$

$$\begin{aligned}\text{Local direct resolution rules for b} = \{&a_2^i \leftarrow\sim a\\ &b_2^i \leftarrow\sim c, \sim b\\ &c_2^i \leftarrow\sim a, \sim c\}\end{aligned}$$

$$\begin{aligned}\text{Local direct resolution rules for c} = \{&a_3^i \leftarrow\sim b, \sim a\\ &b_3^i \leftarrow\sim b\\ &c_3^i \leftarrow\sim a, \sim c\}\end{aligned}$$

What each of these rules states is that under very strict conditions a certain literal x_ι^i can be concluded for local direct resolutions if it is not true in the model and if its direct preconditions are respected.

Let's formalize these definitions:

Definition 43 (Indirect counterfactual program Pcf)**.** *Let P be a normal logic program, X an atom in P which can be proven by* reductio ad absurdum *and ι (iota) an index identifying uniquelly X in the set of literals of P that can be proven by RAA.*

The counterfactual program for indirect OLONs of X is constructed from a copy $Pcf(X^i)$ of P by performing the following operations on $Pcf(X^i)$:

1. *Remove from all rules of $Pcf(X^i)$ the default literal $\sim X$;*
2. *Replace all literals A in all rules with the auxiliary atom A_ι^i;*

Note that the second operation was over literals instead of atoms, as in the definition for direct OLONs.

Additionally, to this counterfactual program we add a copy of the direct OLONS counterfactual program $Pcf(X^d)$ where, on each rule we apply the following changes:

1. *Replace each head A_ι^d with A_ι^i;*
2. *Add to the tail of each rule with head A_ι^i the default $\sim A$.*

We call X_ι^i the head of this counterfactual program. The heads of the rules from which $\sim X$ was removed form the set $Tails_P(X)$.

Definition 44 (Indirect Counterfactual Programs Set)**.** *Let P be a normal logic program, S the set of literals in P which have the potential to be proven by* reductio ad absurdum *and an ordering $\iota \geq 0$ assiciating each literal in S with a number ι.*

The set of all direct counterfactual programs of P regarding all literals in S is the set

$$\beta cf_P^i = \left(\bigcup_{\iota} Pcf(X_\iota^i) \right)$$

with ι being the program's counterfactual index, incremented as a new literal is considered.

Definition 45 (Propagation rules transformation for βcf_P^i)**.** *Let βcf_P^i be the set of all indirect counterfactual programs of the NLP P.*

The propagation rules transformation for each counterfactual program in βcf_P^i consists of the following operation:

- *For each extended literal x_ι^i such that there exists a counterfactual program $Pcf(x^i)$ with head x_κ^i add the rule $x_\iota^i \leftarrow x_\kappa^i$ to the counterfactual program being considered.*

Definition 46 (Rule extension transformation for βcf_P^i)**.** *Let βcf_P^i be the set of all direct counterfactual programs of the NLP P.*

The rule extension transformation for each counterfactual head X_ι^i in βcf_P^i consists of generating the following rule:

$$XOLON \leftarrow X_\iota^i, \sim Tails(X)$$

where $\sim Tails(X)$ corresponds to $\sim t_1, \sim t_2, \ldots, \sim t_n$ if $Tails(X) = t_1, t_2, \ldots, t_n$.

Additionally, for each $XOLON$ generated, the following rules are also to be created:

$$XRAA \leftarrow XOLON$$
$$X \leftarrow XRAA$$

Definition 47 (Indirect OLONs transformation via counterfactual programs)**.** *Let P be a normal logic program.*

The Indirect OLONs transformation via counterfactual programs of P is the revised program P^i which can be generated by the following operations:

1. *Create the counterfactual program set* βcf_P^i*;*
2. *Apply the Propagation Rules transformation and Rule extension transformation on* βcf_P^i*;*
3. *Let* $P^i = \beta cf_P^i$.

We say that the indirect OLONs transformation via counterfactual programs for program P is the auxiliary program P^i.

Theorem 8 (Soundness and completeness of the Indirect OLONs transformation)**.** *From definition 34 on page 44 resulted that for each indirect OLON found, a rule in the form* $a_i \leftarrow\sim Tail^{OLON}(a_i), PC_{a_i}$ *was added to the program, for each possible* a_i.

The program transformation defined in definition 46 instead adds a rule stating $XOLON \leftarrow\sim Tails(X), X_{\mathfrak{t}}^d$.

It holds that the set of rules added by the indirect OLONs transformation via counterfactual programs is equivalent to those of the RAA rule extension property for indirect OLONs.

Proof. We have a two sets of rules which operate on two levels. The first level is that of the counterfactual transformation in itself, and the second is from the extension rules onward.

On the counterfactual level we have, for a given NLP P, several copies of P where each literal which has the potential for being provable by RAA is assumed as false.

It is first important to show that this transformation won't allow atoms to be true if they are not provable by RAA. Consider, for example, that a literal x is true in the Pcf but wasn't in P. Then it became true by assuming its falsity, which means it was provable by *reductio ad absurdum*. Otherwise, assume that x is part of a RAA pattern but is not true by the counterfactual program. Then, x had no support under the counterfactual program, which means that either $PC(x)$ was false or x was not in a RAA pattern.

This point made, all $X_{\mathfrak{t}}^i$ which are true had to be part of $OLONs^i$ and had to had their preconditions true.

This means that in the second level only are true those literals which were part of direct OLONs and had their preconditions true.

Then, the conditions implied by

$$a \leftarrow\sim Tail^{OLON}(a), PCs_a, \text{ given that a is the head of a direct OLON}$$

are satisfied by

$$XOLON \leftarrow\sim Tails(X), X_{\mathfrak{l}}^{i}$$
$$XRAA \leftarrow XOLON$$
$$X \leftarrow XRAA$$

□

Transforming ICONs

Now that we have defined a correct way of detecting OLONs, the only remaining pattern to be transformed are infinite chains over negation. ICONs have the singularity of being infinite programs which in theory means that there is no practical way of calculating their models, in particular because models are calculated on grounded programs. However, a theoretical way of recognizing these patterns can be defined.

ICONs are defined as having two rules supporting the same atom and eventually depending on contradictory tails. As there is no feasible way of detecting the head of the ICON and defining a rule for this head, the only pattern we can detect is the existence of rules with contradictory tails ($\sim X$ and X) and the existence of at least two rules with the same head.

Consider, for example:

$$\begin{aligned} P = \{&\alpha \leftarrow \lambda(1), PC_{\lambda(1)} && \alpha \leftarrow \mu(1), PC_{\mu(1)} \\ &\lambda(1) \leftarrow \dots && \mu(1) \leftarrow \dots \\ &\dots \leftarrow \lambda(n), PC_{\lambda(n)} && \dots \leftarrow \mu(n), PC_{\mu(n)} \\ &\lambda(n) \leftarrow \beta, PC_{\lambda(n+1)} && \mu(n) \leftarrow\sim \beta, PC_{\mu(n+1)}\} \end{aligned}$$

The counterfactual program for P consists of the following transformation:

$$\begin{aligned} P = \{&\alpha_1^c \leftarrow \lambda_1^c(1), PC_{\lambda_1^c(1)} && \alpha_1^c \leftarrow \mu_1^c(1), PC_{\mu_1^c(1)} \\ &\lambda_1^c(1) \leftarrow \dots && \mu_1^c(1) \leftarrow \dots \\ &\dots \leftarrow \lambda_1^c(n), PC_{\lambda_1^c(n)} && \dots \leftarrow \mu_1^c(n), PC_{\mu_1^c(n)} \\ &\lambda_1^c(n) \leftarrow PC_{\lambda_1^c(n+1)} && \mu_1^c(n) \leftarrow PC_{\mu_1^c(n+1)}\} \end{aligned}$$

Note that there is only one counterfactual program generated, where we remove all the contradictory tails. The heads of the rules from which we removed our contradictory tails, if now

are true, should provide support for a certain atom, the head of the ICON. We need now a basic condition for the existence of an ICON, besides that of the program being infinite. The condition is that there are at least two rules with the same head and at least two rules with contradictory tails. Given this condition, and for the previous program, our transformed version of the extension rules for ICONs is:

$$\alpha ICON \leftarrow \alpha_1^c, \lambda_1^c(n), \mu_1^c(n)$$

Note, however, that an infinite number of these rules will exist. Additionally, α is provable by RAA if it is an ICON:

$$\alpha RAA \leftarrow \alpha ICON$$
$$\alpha \leftarrow \alpha RAA$$

Let's formalize these intuitions

Definition 48 (Potential for ICON belonging)**.** *Let P be a normal logic program with l rules in the form $r_k = H \leftarrow A_1, \ldots, A_m, \sim B_1, \ldots, \sim B_n$ ($m, n \geq 0$ and $1 \leq k \leq l$) and X a literal.*

We say that X has the potential for belonging to an infinite chain over negation iff:

$$\begin{aligned} \exists_{r_1, r_2 \in P} : & X = head(r_1) \wedge \\ & X = head(r_2) \wedge \\ & \exists_{r_3, r_4 \in P} : r_3 \text{ and } r_4 \text{ have contradictory tails.} \end{aligned}$$

We say that two tails $Tail_1$ and $Tail_2$ are contradictory if some atom X belongs to $Tail_1$ and $\sim X$ belongs to $Tail_2$.

Definition 49 (ICON counterfactual program Pcf)**.** *Let P be a normal logic program, X a literal in P which has the potential to belong to an ICON.*

The counterfactual program for ICONs of X is constructed from a copy $Pcf(icon)$ of P by performing the following operations on $Pcf(icon)$:

1. *For each set of rules which have contradictory tails, remove the contradictory literals from their tails;*

2. *Replace all literals A in all rules with the auxiliary literal A_1^c.*

We call all literals X_1^c which have the potencial of belonging to an ICON the possible heads of this counterfactual program. The heads of the rules which had contradictory tails form the set $Tails_P(X)$.

Definition 50 (Rule extension transformation for $Pcf(icon)$)**.** *Let $Pcf(icon)$ be the counterfactual program for ICONs of the NLP P.*

The rule extension transformation for each possible head X_1^c in $Pcf(icon)$ consists of generating the following rules for each combination of possible heads and pair of tails:

$$XICON \leftarrow X_1^c, Tails(X)$$

Additionally, for each XICON generated, the following rules are also to be created:

$$XRAA \leftarrow XICON$$
$$X \leftarrow XRAA$$

Definition 51 (ICONs transformation via counterfactual programs)**.** *Let P be a normal logic program.*

The ICONs transformation via counterfactual programs of P is the revised program P^c which can be generated by the following operations:

1. *Create the counterfactual program $Pcf(icon)$ from P;*
2. *Apply the Rule extension transformation on $Pcf(icon)$;*
3. *Let $P^c = Pcf(icon)$.*

We say that the ICONs transformation via counterfactual programs for program P is the auxiliary program P^c.

Theorem 9 (Soundness and completeness of the ICONs transformation)**.** *From definition 34 on page 44 resulted that for each ICON found, two rules in the form*

$$Tail_\lambda^{ICON}(\alpha) \leftarrow PC_{\lambda_{n+1}}, PC_{\mu_{n+1}}$$
$$Tail_\mu^{ICON}(\alpha) \leftarrow PC_{\mu_{n+1}}, PC_{\lambda_{n+1}}$$

were added to the program.

The program transformation defined in definition 50 instead adds several rule stating

$$XICON \leftarrow X_{\iota}^{c}, Tails(X)$$

one for each literal which has the potential for belonging to an ICON.

It holds that the set of rules added by the ICONs transformation via counterfactual programs is equivalent to those of the RAA rule extension property for ICONs.

Proof. For ICONs only one counterfactual program is generated, from which we derive rules from pairs of rules. As there is no feasible way of detecting an ICON, we can only suspect that there is one if at least two rules with the same head exist and two rules with contradictory tails also exist.

This being the case, we say that a certain atom X is true because of being involved in an ICON if both heads became true because of the contradiction having been removed. Additionally, as we are now creating dependency directly on the tail of the ICON, the preconditions don't need to be tested in particular – if they are true they will provide support for the ICON. Otherwise they won't.

Furthermore, let's suppose that some atom X was not part of an ICON and was false in the model. If X comes true from this transformation, then a chain of support had to exist between X and its tails. This is however proves that X was part of an ICON. Let's now suppose that X was part of an ICON. The only way for X not to be concludable by the counterfactual program transformation for ICONs, and not be part of the model, is if any of its preconditions are false, now that the contradictory tails have been removed. This not being the case, X has to be true.

Therefore, the transformation is equivalent to the rule extension for ICONs. □

4.3.2 Transformational Semantics for the revised stable models

Given the previous transformations we can now define the Γ^r operator for revised stable models:

Definition 52 (Γ^r operator)**.** *Let P be a normal logic program and I a two-valued interpretation.*

The revised program P^r is the following sequence of operations:

1. *Generate the Direct OLONs transformed program P^d;*
2. *Generate the Indirect OLONs transformed program P^i;*
3. *Generate the ICONs transformed program P^c;*
4. *Let $P^r = P^d \cup P^i \cup P^c \cup P$.*

On P^r apply $\Gamma(I)$ and remove all counterfactual auxiliary literals from the resulting set F.

We say that $\Gamma^r(I) = \Gamma_{P^r}(I) = F$

We can now provide an alternative definition of the revised stable models semantics based on this operator:

Definition 53 (Revised Stable Models and Semantics redefinition)**.** *Let P be a normal logic program and I a two-valued interpretation.*

We say that I is a revised stable model of P iff $\Gamma^r(I) = I$.

Theorem 10 (Soundness and completeness regarding the revised stable models semantics)**.** *Having firstly defined the RAA rule extension property and the equivalence between this property and the counterfactual programs defined in the previous section, we can now show that a program transformation based on the counterfactual programs is equivalent to the revised stable models.*

Proof. As by theorems 7, 8 and 9 all the counterfactual programs generated are equivalent to the extension rules added by Σ_P, and because Γ^r employs these transformations, by theorem 5 I is a revised stable model of P iff $\Gamma^r(I) = I$. □

4.3.3 Transformational Semantics for the revised well-founded semantics

Following the same approach Baral and Subrahmanian used in [5], we now define Γ^{r^2} and the rWFS based on this operator.

Definition 54 (Γ^{r^2})**.** *Let P be a NLP and I an interpretation.*

The Γ^{r^2} is defined as being two applications of the Γ^r operator:

$$\Gamma_P^{r^2}(I) = \Gamma_P^r(\Gamma_P^r(I))$$

At this point we can prove the only property of the rWFS for which a proof was still missing, that the revised well-founded semantics can be calculated by a monotonous and continuous operator.

Theorem 11 (Γ^{r^2} is a monotonous and continuous operator)**.** *It holds that Γ^{r^2} is a monotonous and continuous operator.*

Proof. As for any normal logic program P, $\Gamma_P^r = \Gamma_{P^r}$, where P^r is the original program P transformed according to the Γ^r operator, we have:

$$\Gamma_P^{r^2} = \Gamma_P^r(\Gamma_P^r) \Leftrightarrow \tag{4.2}$$

$$\Leftrightarrow \Gamma_P^r(\Gamma_P^r) = \Gamma_{P^r}(\Gamma_{P^r}) \Leftrightarrow \tag{4.3}$$

$$\Leftrightarrow \Gamma_{P^r}(\Gamma_{P^r}) = \Gamma_{P^r}^2 \tag{4.4}$$

However, Γ^2 applied on any normal logic program is a monotonous and continuous operator, so the theorem is trivially true. □

Definition 55 (Revised Well-Founded Model and Semantics alternative definition)**.** *Let P be a NLP.*

The Revised Well-Founded Semantics of program P is the tuple $\langle \mathcal{T}_{rWFM}, \mathcal{TU}_{rWFM} \rangle$, with $\mathcal{T}_{rWFM}$ corresponding to the true literals according to the revised well-founded semantics and $\mathcal{TU}_{rWFM}$ being the true or undefined literals according to the revised well-founded semantics.

The $\mathcal{T}_{rWFM}$ set can be calculated as being the least fixed point of Γ^{r^2} operator starting with the empty interpretation, i.e., $\mathcal{T}_{rWFM} = \Gamma_P^{r^2\uparrow\omega}$, with ω the least limit ordinal.

The $\mathcal{TU}_{rWFM}$ set can be calculated as being the next iteration of Γ_P^r, starting from the $\mathcal{T}_{WFM}$ set, i.e., $\mathcal{TU}_{rWFM} = \Gamma_P^r(\mathcal{T}_{rWFM})$.

Theorem 12 (Soundness and completeness regarding the revised well-founded semantics)**.** *Following the same approach of the redefinition of the revised stable models, an alternative definition of the revised well-founded semantics can be presented, based on two applications of the Γ operator.*

Proof. As by theorems 7, 8 and 9 all the counterfactual programs generated are equivalent to the extension rules added by Σ_P, and because Γ^{r^2} employs these transformations, by theorem 6 $I = \langle \mathcal{T}_{rWFM}, \mathcal{TU}_{rWFM} \rangle$ is the revised well-founded model of P iff $\mathcal{T}_{rWFM} = \Gamma_P^{r^2\uparrow\omega}$, with ω the least limit ordinal, and $\mathcal{TU}_{rWFM} = \Gamma_P^r(\mathcal{T}_{rWFM})$. □

Additionally, we can now define the revised partial stable models as being non-minimal fixed points of Γ^{r^2} operator:

Definition 56 (Revised Partial Stable Models)**.** *Let P be a normal logic program and I a three-valued interpretation.*

$I = \mathcal{T} \cup \sim \mathcal{F}$ is a revised partial stable model of P iff:

- $\mathcal{T} = \Gamma^{r^2}(\mathcal{T})$
- $\mathcal{T} \subseteq \Gamma^r(\mathcal{T})$
- $\mathcal{F} = \mathcal{H}_P \setminus \Gamma^r(\mathcal{T})$

Revised partial stable models can be represented in a lattice, just like its non-RAA counterpart, where each node corresponds to a fixed point of Γ^{r^2}*. The minimal fixed point is the rWFM.*

A thorough study of all the complexity issues of the revised well-founded semantics is out of the scope of this thesis. However, a brief account on this subject can be provided, thus enabling us to say that the calculation of the rWFM has polynomial complexity.

Theorem 13 (The calculation of the rWFM has polynomial complexity)**.** *Let P be a normal logic program and $rWFM_P = \langle \mathcal{T}^r, \mathcal{TU}^r \rangle$ the revised well-founded model of a normal logic program P.*

It holds that the complexity of generating the revised well-founded model is polynomial.

Proof. Consider $WFM(P)$ the complexity of calculating the well-founded model of a normal logic program P. We know that $WFM(P)$ is polynomial.

Let's also consider $tr(P)$ the complexity of generating the program transformations required for counterfactual programs of the normal logic program P. We know that this process includes the generation of four copies of the original program for each literal which has the potential of being provable by RAA.

Let's call $copy(P)$ the generation of a copy of program P. Generating a copy of a program is a deterministic procedure which, in our case, includes renaming all literals to their counterfactual versions and removing some default literals. These operations, however, can be applied as the copies of the rules of the program are being generated, meaning that we have $O(copy(P)) = n$, with n being the number of literals which had the potential for being provable by RAA.

The transforming procedure will need to perform four copies of the program: one for direct OLONs, two for indirect OLONs and one for ICONs. Therefore we have that:

$$O(rWFS_P) = \tag{4.5}$$
$$= O(WFM(tr(P))) = \tag{4.6}$$
$$= O(WFM(4 \times copy(P))) = \tag{4.7}$$
$$= O(WFM(4 \times n)) = \tag{4.8}$$
$$= \text{polynomial complexity} \tag{4.9}$$

□

4.4 Examples

In this section we show the transformation of two simple programs, essentially to illustrate the counterfactual programs transformation. Given the complexity of the transformation, we include only two examples and leave the rest of the results to the appendix A.

Example 15 (OLON with inner OLON[7])**.**

$$\begin{aligned} P = \{ & a \leftarrow x \\ & x \leftarrow y, \sim x \\ & y \leftarrow \sim a \} \end{aligned}$$

In this program we have two literals which have the potential for being provable by RAA, a and x. We will create counterfactual programs for both literals, both direct and indirect because we don't know for sure what kind of OLONs we may be dealing with here. We start with the counterfactual programs for direct OLONS:

$$\begin{aligned} Pcf(a^d) = \{ & a_1^d \leftarrow x_1^d \\ & x_1^d \leftarrow y_1^d, \sim x \\ & y_1^d \leftarrow \\ & x_1^d \leftarrow x_2^d \} \\ & aOLON \leftarrow a_1^d, \sim y \\ & xOLON \leftarrow x_2^d \end{aligned}$$

$$\begin{aligned} Pcf(x^d) = \{ & a_2^d \leftarrow x_2^d \\ & x_2^d \leftarrow y_2^d \\ & y_2^d \leftarrow \sim a \\ & a_2^d \leftarrow a_1^d \} \end{aligned}$$

We also generate the counterfactual programs for indirect OLONs:

$$\begin{aligned} Pcf(a^i) = \{ & a_1^i \leftarrow x_1^i \\ & x_1^i \leftarrow y_1^i, \sim x_1^i \\ & y_1^i \leftarrow \\ & a_1^i \leftarrow x_1^i, \sim a \\ & x_1^i \leftarrow y_1^i, \sim x \\ & y_1^i \leftarrow \sim y \\ & x_1^i \leftarrow x_2^i \} \\ & aOLON \leftarrow a_1^i, \sim y \\ & xOLON \leftarrow x_2^i \end{aligned}$$

$$\begin{aligned} Pcf(x^i) = \{ & a_2^i \leftarrow x_2^i \\ & x_2^i \leftarrow y_2^i \\ & y_2^i \leftarrow \sim a_2^i \\ & a_2^i \leftarrow x_2^i, \sim a \\ & x_2^i \leftarrow y_2^i, \sim x \\ & y_2^i \leftarrow \sim a, \sim y \\ & a_2^i \leftarrow a_1^i \} \end{aligned}$$

[7] This example is due to Sérgio Lopes

And finally the rules which connect with the original literals:

$$\begin{aligned} \textit{Extension Rules} = \{ & \textit{aRAA} \leftarrow \textit{aOLON} \\ & \textit{xRAA} \leftarrow \textit{xOLON} \\ & a \leftarrow \textit{aRAA} \\ & x \leftarrow \textit{xRAA}\} \end{aligned}$$

The only stable model of the program $P^r = P \cup Pcf(a^d) \cup Pcf(x^d) \cup Pcf(a^i) \cup Pcf(x^i) \cup$ *Extension Rules is* $rSM_1 = \{a, aRAA, aOLON, x_1^i, y_1^i, a_1^i, a_2^i, a_1^d, x_1^d, y_1^d\}$*, from which we remove all the extended literals and end up with* $rSM_1 = \{a\}$*.*

Example 16 (Indirect OLON)**.**

$$\begin{aligned} P = \{ & a \leftarrow\sim b \\ & b \leftarrow\sim c \\ & c \leftarrow\sim a\} \end{aligned}$$

In this program, all literals have the potential for being proven by RAA so we generate all possible counterfactual programs. We start with the counterfactual programs for direct OLONS:

$$\begin{aligned} Pcf(a^d) = \{ & a_1^d \leftarrow\sim b \\ & b_1^d \leftarrow\sim c, \\ & c_1^d \leftarrow \\ & b_1^d \leftarrow b_2^d \\ & c_1^d \leftarrow c_3^d\} \end{aligned} \qquad \begin{aligned} Pcf(b^d) = \{ & a_2^d \leftarrow \\ & b_2^d \leftarrow\sim c \\ & c_2^d \leftarrow\sim a \\ & a_2^d \leftarrow a_1^d \\ & c_2^d \leftarrow c_3^d\} \end{aligned} \qquad \begin{aligned} Pcf(c^d) = \{ & a_3^d \leftarrow\sim b \\ & b_3^d \leftarrow \\ & c_3^d \leftarrow\sim a \\ & b_3^d \leftarrow b_2^d \\ & a_3^d \leftarrow a_1^d\} \end{aligned}$$

$$\begin{aligned} & \textit{aOLON} \leftarrow a_1^d, \sim c \\ & \textit{bOLON} \leftarrow b_2^d, \sim a \\ & \textit{cOLON} \leftarrow c_3^d, \sim b \end{aligned}$$

We also generate the counterfactual programs for indirect OLONs:

$$
\begin{aligned}
Pcf(a^i) = \{ & a_1^i \leftarrow\sim b_1^i \\
& b_1^i \leftarrow\sim c_1^i \\
& c_1^i \leftarrow \\
& a_1^i \leftarrow\sim b, \sim a \\
& b_1^i \leftarrow\sim c, \sim b \\
& c_1^i \leftarrow\sim c \\
& b_1^i \leftarrow b_2^i \\
& c_1^i \leftarrow c_3^i \}
\end{aligned}
\qquad
\begin{aligned}
Pcf(b^i) = \{ & a_2^i \leftarrow \\
& b_2^i \leftarrow\sim c_2^i \\
& c_2^i \leftarrow\sim a_2^i \\
& a_2^i \leftarrow\sim a \\
& b_2^i \leftarrow\sim c, \sim b \\
& c_2^i \leftarrow\sim a, \sim c \\
& a_2^i \leftarrow a_1^i \\
& c_2^i \leftarrow c_3^i \}
\end{aligned}
\qquad
\begin{aligned}
Pcf(c^i) = \{ & a_3^i \leftarrow\sim b_3^i \\
& b_3^i \leftarrow \\
& c_3^i \leftarrow\sim a_3^i \\
& a_3^i \leftarrow\sim b, \sim a \\
& b_3^i \leftarrow\sim b \\
& c_3^i \leftarrow\sim a, \sim c \\
& b_3^i \leftarrow b_2^i \\
& a_3^i \leftarrow a_1^i \}
\end{aligned}
$$

$$
\begin{aligned}
& aOLON \leftarrow a_1^i, \sim c \\
& bOLON \leftarrow b_2^i, \sim a \\
& cOLON \leftarrow c_3^i, \sim b
\end{aligned}
$$

And finally the rules which connect with the original literals:

$$
\begin{aligned}
Extension\ Rules = \{ & aRAA \leftarrow aOLON \\
& bRAA \leftarrow bOLON \\
& cRAA \leftarrow cOLON \\
& a \leftarrow aRAA \\
& b \leftarrow bRAA \\
& c \leftarrow cRAA \}
\end{aligned}
$$

This program will now have three rSMs:

$$
\begin{aligned}
rSM_1 &= \{a, b, aRAA, aOLON, b_2^d, a_3^i, \ldots\} = \{a, b\} \\
rSM_2 &= \{b, c, bRAA, bOLON, c_3^d, b_1^i, \ldots\} = \{b, c\} \\
rSM_3 &= \{c, a, cRAA, cOLON, c_2^i, a_1^d, \ldots\} = \{c, a\}
\end{aligned}
$$

4.5 Concluding Remarks

Having first identified the property of rule extension for revised stable models, we defined a theoretical operator, one that if it was possible to find out all RAA patterns of a program, it would add rules to the program so that RAA patterns would enjoy classical support. From this idea we created a program transformation that followed the same principle, but skipped the step of detecting OLONs / ICONs by creating counterfactual programs. This is what enabled

us to define Γ^r operator, which can be used to calculate the revised stable models and the revised well-founded model of a normal logic program. We also proved the last of the properties set in section 3.1 – that the revised well-founded semantics is definable by a monotonous and continuous operator.

[This page was intentionally left blank]

Chapter 5

Concluding Remarks and Future Work

Contents

Where we present the open issues that arose with the introduction of the principle of Reductio Ad Absurdum *in the Well-Founded Semantics, and outline some possible next steps in this thread of research.*

5.1 Concluding Remarks and Future Work

With the conclusion of this step in this *revised* family of semantics, some final remarks should be pointed out.

First, that the definition of this semantics followed a quite different path from the previous semantics in this family of revised semantics. While the revised stable models were defined because of the problems the stable models had, we started this thesis preciselly by pointing out that the well-founded semantics had no problems whatsoever, in our view. Not only are they always capable of determining a model for a given normal logic program, as the fact that it doesn't deal correctly with certain program patterns should be viewed more like a feature than a fault. This allows us to point out that the rWFS is not a substitute of the WFS, but rather a counterpart, which extends it in a three-valued setting by allowing another form of reasoning – reasoning by absurdity.

On the other hand, the rWFS ended up filling a gap which had been opended with the introduction of the rSMs – that of the transition between a three-valued context to a two-valued one *with* RAA support. By introducing the revised well-founded semantics, we made it possible to have this transition defined, namelly through the introduction of the revised partial stable models. The fact that this transition is now well defined brings both semantics closer and opens new possibilities of finding more relations between them, much like the same way several relations were eventually found between the stable models and the well-founded semantics.

Additionally, there is the issue of using the principle of *reductio ad absurdum* as a form of support. While this had already been done in the revised stable models semantics, it was now implemented in the context of a three-valued semantics. It not only proved that the principle is a valid one for reasoning in nonmonotonic contexts, as it proved it maintains its validity when we introduce another truth value.

Finally, it's worth pointing out that the relation between these two semantics and their non-RAA counterparts is now much closer, mainly because of the definition of the Γ^r operator. Firstly it is of great importance the fact that the property of RAA rule extension was defined, as this was a missing piece in defining a program transformation whose stable models would correspond to the revised stable models. Additionally, the fact that we used an extension of the already well-known and used Γ operator, allows us to present these revised semantics as the original ones with a plug-in called *reductio ad absurdum*. This, indeed, shows how close they are, the only difference being the additinal form of support present in the *revised* family of semantics.

We believe that these results are of great importance to the continuing progress of this field of research, as it opens new possibilities and creates new connections with other semantics. We'll analyse now with some detail several possible paths for future work.

One of the issues which we speak of in the motivation for this thesis, in page 2, is the fact that

in some cases, some conclusions could be derived from literals which already were undefined. One such example is the following program:

$$\begin{aligned} P = \{ & a \leftarrow\sim b \\ & b \leftarrow\sim a \\ & c \leftarrow a \\ & c \leftarrow b\} \end{aligned}$$

It is known that $WFM_P = rWFM_P = \langle\{\},\{a,b,c\}\rangle$. However, independently of a and b being undefined, one could argue that this value of undefinedness should not propagate to c and that it is impossible for c to ever false in any world. Therefore it should be true in the WFM.

The definition of the reasoning behind this result is of interest when comparing it with the rWFS and the WFS. Apparently a skepticalness ordering could be defined here, where the WFS, being the most skeptical, would also be the least expressive. Then, in terms of skepticalness would come the rWFS and then this semantics (should it be defined). Under this new semantics the only literals to have the value of undefined would be those present in choices and undefinedness would not necessarilly propagate itself.

Still with respect to other three-valued formalisms would be the relation between the WFS, the rWFS and the O-Semantics [29]. The O-Semantics is defined as an enlargement of the well-founded semantics which gives meaning to the adding of Closed World Assumptions to further add some conclusions to the well-founded semantics. Consider the following program:

$$\begin{aligned} P = \{ & a \leftarrow\sim a \\ & c \leftarrow\sim a\} \end{aligned}$$

Under the WFS, nothing is true and everything is undefined. Under the rWFS, a is true and therefore c is false. Under the O-Semantics, the authors claim that c should be false because $\sim a$ will never be true in any model of P. However, they allow a to remain undefined. It's easy to see that the O-Semantics is somewhat in between the WFS and the rWFS in terms of skepticalness, and so a formal study of this relation is not only important as it is relevant in integrating the rWFS even more in the context of three-valued formalisms.

Still regarding three-valued formalisms, the extension of the rWFS into its extended counterpart (rWFSX) is also a relevant issue. This would allow the definition of the relation of the principle of RAA with a second form of negation.

On a more theoretical side is the study of the relation of the rWFS and the rSMs with non-monotonic reasoning formalisms. Being the principle of *reductio ad absurdum* a principle so extensivelly used in common sense reasoning as well as in formal reasoning it is interesting to see how it relates to NMR formalisms. The study of whether or not a close relation between the rWFS and NMR formalisms exists would be a consolidating result for this semantics. It might

even generate new results for the relation between the WFS and the rWFS, as well as provide a broader study of the application of the RAA principle.

Another study that is missing is a thorough complexity study. In the previous chapter we provided a small account on the complexity of calculating the revised well-founded model based on the Γ^r operator. However more complexity issues are to be analized and this study is an important one. Still regarding complexity issues, there is also the need to study the gain associated with using the revised well-founded semantics to simplify the calculation of the revised stable models by first calculating the revised well-founded model and then only using the revised stable models to calculate the undefined literals. This would not only be an important implementation work but also relevant in finding out what were the associated gains when using this strategy.

Finally, it would be important to study a more exhaustive definition of all this family of semantics, without ever recurring to operational concepts like loops, and keeping the tradition in this field, by only stating the principles it should obey. Part of this has already been accomplished by the definition of the RAA rule extension property, which adds a rule which states the RAA reasoning behind RAA patterns. However a purelly declarative definition of all these semantics is a hard task because part of the principle we are using here is inherently operational.

So far we have succeeded in applying this principle to a two-valued semantics, capable of extending the stable models, and to a three-valued semantics, capable of extending the well-founded semantics. This last extension provided a way of viewing a given situation skeptically, while still making use of the principle of *reductio ad absurdum* to derive conclusions.

Even the strange rules employed by Anthony Kiedis of the Red Hot Chilli Peppers in [27], should they need to make use of support by absurdity, are now formally defined under the Revised Well-Founded Semantics.

Bibliography

[1] J. Alcântara. *Semânticas disjuntivas para programação em lógica estendida*. PhD thesis, Departamento de Informática, Faculdade de Ciências e Tecnologia, UniversidadeNova de Lisboa, Portugal, 2006.

[2] J. J. Alferes. *Semantics of Logic Programs with Explicit Negation*. PhD thesis, Departamento de Informática, Faculdade de Ciências e Tecnologia, UniversidadeNova de Lisboa, Portugal, 1993.

[3] G. Antoniou. *Nonmonotonic Reasoning*. Artificial Intelligence Series. The MIT Press, Cambridge, Massachusets, USA, April 1997.

[4] J. N. Aparício. *Logic Programming: A Tool for Reasoning*. PhD thesis, Departamento de Informática, Faculdade de Ciências e Tecnologia, UniversidadeNova de Lisboa, Portugal, 1993.

[5] C. R. Baral and V. S. Subrahmanian. Dualities between alternative semantics for logic programming and nonmonotonicreasoning. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, pages 69–86, College Park, MD, USA, 1991. MIT Press.

[6] N. Bidoit and C. Froidevaux. General logic databases and programs: default logic semantics and stratification. *Journal of Information and Computation*, 1988.

[7] A. Colmerauer, H. Kanoui, R. Pasero, and P. Roussel. Un système de communication homme-machine en français. Technical report, Groupe de Intelligence Artificielle, Universite de Aix-Marseille II, 1973.

[8] A. Colmerauer and P. Roussel. The birth of prolog. In T. J. Bergin and R. G. Gibson, editors, *History of programming languages—II*, ACM SIGPLAN Notices, pages 331–367, New York, NY, USA, 1996. ACM Press/Addison-Wesley.

[9] C. V. Damásio and L. M. Pereira. A paraconsistent semantics detecting contradition support. In J. Dix, U. Furbach, and A. Nerode, editors, *4th International Conference on Logic Programming and Nonmonotonic Reasoning*, LNAI, pages 224–243. Springer, 1997.

[10] P. M. Dung. On the relation between stable and well-founded semantics of logic programs. *Theoretical Computer Science: Logic, semantics and theory of programming*, 2:75–78, 1990.

[11] F. Fages. Consistency of clark's completion and existence of stable models. *Methods of Logic in Computer Science*, 38(3):620–650, July 1991.

[12] M. C. Fitting. Well-founded semantics, generalized. In *International Symposium on Logic Programming*, pages 71–84, 1991.

[13] M. Gelfond. On stratified autoepistemic theories. In M. Kaufman, editor, *AAAI'87*, pages 207–211, 1987.

[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, USA, 1988. The MIT Press.

[15] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*, pages 579–597. MIT Press, 1990.

[16] R. Kowalski. Algorithm = logic + control. *Commun. ACM*, 22(7):424–436, 1979.

[17] R. A. Kowalski. Predicate logic as a programming language. In *Proceedings of IFIP'74*, pages 569–574. North Holland Publishing Company, 1974.

[18] J. A. Leite. *Evolving Knowledge Bases – Specification and Semantics*, volume 81 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2003.

[19] W. Marek and M. Truszczynski. Autoepistemic logics. *Journal of the ACM*, 38(3):588–619, 1991.

[20] J. McCarthy. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London, 1959. Her Majesty's Stationary Office.

[21] J. McCarthy and P. Hayes. Some philosofical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[22] J. McCarthy, M. Minsky, N. Rochester, and C. Shannon. A proposal for the dartmouth summer research project on artificial intelligence, August 1955.

[23] D. McDermott. A critique of pure reason. *Computational Intelligence*, 3:151–237, 1987.

[24] M. Minsky. A framework for representing knowledge. Technical report, MIT Artificial Intelligence Laboratory, 1974.

[25] R. Moore. Semantics considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[26] N. J. Nilsson. Logic and artificial intelligence. MIT Workshop on Foundations of Artificial Intelligence, June 1987.

[27] R. H. C. Peppers. Around the world. Californication, Warner Bros., 1999.

[28] L. M. Pereira and J. J. Alferes. Well-founded semantics for logic programs with explicit negation. In B. Neumann, editor, *European Conference on Artificial Intelligence*, pages 102–106. John Wiley & Sons, 1992.

[29] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Adding closed world assumptions to well-founded semantics. *Theoretical Computer Science*, 122(1–2):49–68, 1994.

[30] L. M. Pereira and A. M. Pinto. Revised stable models – a new semantics for logic programs. In *Proceedings of the convegno Italiano di Logica Computazionale (CILC'04)*, Parma, Italy, July 2004.

[31] L. M. Pereira and A. M. Pinto. Revised stable models – a semantics for logic programs. In C. Bento, A. Cardoso, and G. Dias, editors, *Progress in Artificial Intelligence, Procs. 12th Portuguese Intl.Conf. onArtificial Intelligence (EPIA'05)*, pages 29–42. Springer, December 2005.

[32] A. M. Pinto. Explorations in revised stable models – a new semantics for logic programs. Master's thesis, Departamento de Informática, Faculdade de Ciências e Tecnologia,Universidade Nova de Lisboa, Portugal, 2005.

[33] H. Przymusinska and T. C. Przymusinski. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Techniques in Artificial Intelligence, a Sourcebook*, pages 321–367. North Holland, 1990.

[34] T. C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 1989.

[35] P. Rao, K. Sagonas, T. Swift, D. S. Warren, and J. Freire. Xsb: A system for efficiently computing wfs. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 430–440, Berlin, July 1997.

[36] R. Reiter. A logic for defalut reasoning. *Artificial Intelligence*, 13:68–93, 1980.

[37] V. S. Subrahmanian, D. Nau, and C. Vago. Wfs + branch and bound = stable models. Technical report, Department of Computer Science, University of MAryland at College Park, 1992.

[38] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, October 1976.

[39] A. van Gelder. The alternating fixpoint of logic programs with negation. In *Proceedings of the 8th ACM Symposium on Principles of Database Systems*, pages 1–10, 1989.

[40] A. van Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.

[41] XSB-Prolog. The xsb logic programming system, version 2.5. Available at `http://xsb.sourceforge.net/`, 2002.

[This page was intentionally left blank]

Appendix A

Examples of Calculation of the rWFS and the rSMs

Contents

Where we show the results of the calculation of the revised stable models and the revised well-founded model of several programs by the implementation discussed in the next appendix.

A.1 Calculation Examples of rWFS and the rSMs

Here we present some calculation examples of programs with and without RAA patterns as provided by the implementation presented in the next chapter. These results are already treated to improve readability and all counterfactual auxiliary literals were removed.

A.1.1 Program 1

$$
\begin{aligned}
P = \{ & a \leftarrow x \\
& x \leftarrow y \\
& y \leftarrow\sim a \\
& y \leftarrow\sim b\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{a,x,y\} \\
& rWFM = \langle\{a,x,y\},\{a,x,y\}\rangle
\end{aligned}
$$

A.1.2 Program 2

$$
\begin{aligned}
P = \{ & a \leftarrow x,y \\
& x \leftarrow z \\
& z \leftarrow\sim a \\
& y \leftarrow w \\
& w \leftarrow\sim a\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{a\} \\
& rWFM = \langle\{a\},\{a\}\rangle
\end{aligned}
$$

A.1.3 Program 3

$$
\begin{aligned}
P = \{ & a \leftarrow x \\
& x \leftarrow\sim a, y \\
& y \leftarrow\sim b \\
& b \leftarrow z \\
& z \leftarrow\sim b, w \\
& w \leftarrow\sim a\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{a, y\} \\
& rSM_2 = \{b, w\} \\
& rWFM = \langle \{\}, \{a, y, b, w, x, z\} \rangle
\end{aligned}
$$

A.1.4 Program 4

$$
\begin{aligned}
P = \{ & a \leftarrow\sim a \\
& b \leftarrow\sim a, \sim b\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{a\} \\
& rWFM = \langle \{a\}, \{a\} \rangle
\end{aligned}
$$

A.1.5 Program 5

$$
\begin{aligned}
P = \{ & a \leftarrow\sim b, \sim c \\
& b \leftarrow\sim a, \sim c \\
& c \leftarrow\sim a, \sim b\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{a\} \\
& rSM_2 = \{b\} \\
& rSM_3 = \{c\} \\
& rWFM = \langle \{\}, \{a, b, c\} \rangle
\end{aligned}
$$

A.1.6 Program 6

$$
\begin{aligned}
P = \{ & a \leftarrow\sim b \\
& b \leftarrow\sim c, x \\
& c \leftarrow\sim a \\
& x \leftarrow\sim x, a\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{a, b, x\} \\
& rSM_2 = \{b, c\} \\
& rSM_3 = \{c, a, x\} \\
& rWFM = \langle \{\}, \{a, b, c, x\} \rangle
\end{aligned}
$$

A.1.7 Program 7

$$
\begin{aligned}
P = \{ & a \leftarrow x \\
& x \leftarrow\sim a \\
& b \leftarrow\sim a \\
& c \leftarrow\sim b \\
& d \leftarrow\sim c\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{a, c\} \\
& rWFM = \langle \{a, c\}, \{a, c\} \rangle
\end{aligned}
$$

A.1.8 Program 8

$$
\begin{aligned}
P = \{ & b \leftarrow y \\
& y \leftarrow\sim a, c \\
& c \leftarrow a \\
& a \leftarrow x \\
& x \leftarrow\sim b\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{b\} \\
& rSM_2 = \{a, c, x\} \\
& rWFM = \langle \{\}, \{b, a, c, x, y\} \rangle
\end{aligned}
$$

A.1.9 Program 9

$$
\begin{aligned}
P = \{ & a \leftarrow x \\
& x \leftarrow y \\
& y \leftarrow \sim a \\
& y \leftarrow \sim x\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{a,x\} \\
& rWFM = \langle \{a,x\}, \{a,x\} \rangle
\end{aligned}
$$

A.1.10 Program 10

$$
\begin{aligned}
P = \{ & a \leftarrow \sim b \\
& b \leftarrow \sim a \\
& c \leftarrow a, \sim c \\
& x \leftarrow \sim y \\
& y \leftarrow \sim x \\
& z \leftarrow x, \sim z\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{b,y\} \\
& rSM_2 = \{b,x,z\} \\
& rSM_3 = \{a,c,y\} \\
& rSM_4 = \{a,c,x,z\} \\
& rWFM = \langle \{\}, \{a,c,x,z,b,y\} \rangle
\end{aligned}
$$

A.1.11 Program 11

$$
\begin{aligned}
P = \{ & a \leftarrow x \\
& x \leftarrow y, \sim x \\
& y \leftarrow \sim a\}
\end{aligned}
$$

$$
\begin{aligned}
& rSM_1 = \{a\} \\
& rWFM = \langle \{a\}, \{a\} \rangle
\end{aligned}
$$

A.1.12 Program 12

$$P = \{a \leftarrow x$$
$$x \leftarrow y, \sim a$$
$$y \leftarrow \sim a\}$$

$$rSM_1 = \{a\}$$
$$rWFM = \langle \{a\}, \{a\} \rangle$$

A.1.13 Program 13

$$P = \{c \leftarrow a, \sim c$$
$$a \leftarrow \sim b$$
$$b \leftarrow \sim a\}$$

$$rSM_1 = \{a, c\}$$
$$rSM_2 = \{b\}$$
$$rWFM = \langle \{\}, \{a, c, b\} \rangle$$

A.1.14 Program 14

$$P = \{a \leftarrow \sim b, \sim a$$
$$b \leftarrow \sim a, \sim b\}$$

$$rSM_1 = \{a\}$$
$$rSM_2 = \{b\}$$
$$rWFM = \langle \{\}, \{a, b\} \rangle$$

A.1.15 Program 15

$$P = \{a \leftarrow\sim b$$
$$b \leftarrow\sim c$$
$$c \leftarrow\sim a\}$$

$$rSM_1 = \{a,b\}$$
$$rSM_2 = \{b,c\}$$
$$rSM_3 = \{c,a\}$$
$$rWFM = \langle\{\},\{a,b,c\}\rangle$$

A.1.16 Program 16

$$P = \{a \leftarrow\sim a\}$$

$$rSM_1 = \{a\}$$
$$rWFM = \langle\{a\},\{a\}\rangle$$

A.1.17 Program 17

$$P = \{a \leftarrow\sim a$$
$$b \leftarrow\sim a\}$$

$$rSM_1 = \{a\}$$
$$rWFM = \langle\{a\},\{a\}\rangle$$

A.1.18 Program 18

$$P = \{a \leftarrow b, x$$
$$b \leftarrow\sim b$$
$$x \leftarrow\sim a\}$$

$$rSM_1 = \{a, b\}$$
$$rWFM = \langle\{a, b\}, \{a, b\}\rangle$$

A.1.19 Program 19

$$P = \{a \leftarrow\sim a, \sim b$$
$$b \leftarrow\sim b, d$$
$$d \leftarrow\sim a\}$$

$$rSM_1 = \{a\}$$
$$rSM_2 = \{b, d\}$$
$$rWFM = \langle\{\}, \{a, b, d\}\rangle$$

A.1.20 Program 20

$$P = \{y \leftarrow\sim a$$
$$a \leftarrow\sim b$$
$$b \leftarrow\sim c\}$$

$$rSM_1 = \{y, b\}$$
$$rWFM = \langle\{y, b\}, \{y, b\}\rangle$$

Appendix B

Implementation Source Code

Contents

Where we present the implementation source code of the Γ^r *operator translator and the rWFM calculator*

B.1 Γ^r operator translator

The Γ^r operator translator basically processes a file with a normal logic program and generates the program transformation for direct and indirect OLONs. This transformed program can then be called from within a stable model calculator and its stable models will correspond to the revised stable models of the original program.

This part of the implementation is divided in several modules.

B.1.1 translator.P

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Includes



:- import concat_atom/2 from string.
:- import term_to_atom/2 from string.
:- import term_to_codes/2 from string.
:- import atom_to_term/2 from string.
:- import append/3 from basics.
:- import length/2 from basics.
:- import member/2 from basics.
:- import flatten/2 from basics.

:- dynamic pos/1, neg/1, raa/1, tails/2.

:- consult('raa2.P').
:- consult('direct.P').
:- consult('indirect.P').
:- consult('tails.P').
:- consult('preprocess.P').


%%%%%%%%%%%%%%%% READING THE MAIN PROGRAM

consultRaa(File):-
clearall,
generateRAAset(File),
raa(RAASet),
generateTailSet(RAASet,File),
```

```
tell('raa.P'),
generateProgramCopy(File),
generateDirectCFP(RAASet,File),
generateIndirectCFP(RAASet,File),
generateAuxRules(RAASet),
told,
consult('raa.P').



%%%%%%%%%%%%%%%%%%%%%% auxiliary rules

generateAuxRules([H|T]):-
concat_atom([H,raa],HeadRAA),
translateLiteral(H,DirectH,direct,H),
translateLiteral(H,IndirectH,indirect,H),
tails(H,TailSet),
generateTailNegation(NegatedTail,TailSet),
concat_atom([NegatedTail,',',IndirectH],IndirectH2),
concat_atom([NegatedTail,',',DirectH],DirectH2),
saveTable(DirectH),
saveRule(HeadRAA,DirectH2),
saveRule(HeadRAA,IndirectH2),
saveRule(H,HeadRAA),
generateAuxRules(T).

generateAuxRules([]).



%%%%%%%%%%%%%%%%%%% generate tnots for tails
generateTailNegation(NegatedTail,[H|T]):-
concat_atom([tnot,'(',H,')'],OneTail),
generateTailNegation(SubNegatedTail,T),
concat_atom([OneTail,',',SubNegatedTail],PreNegatedTail),
NegatedTail = PreNegatedTail.


generateTailNegation(NegatedTail,[]):-
NegatedTail = true.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% generates all direct cfps
generateDirectCFP([H|T],File):-
generateDirectOLONSet(H,File),
generateDirectCFP(T,File).

generateDirectCFP(Atom,File):-
generateDirectOLONSet(Atom,File).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% generates all indirect cfps
generateIndirectCFP([H|T],File):-
generateIndirectOLONSet(H,File),
generateIndirectCFP(T,File).

generateIndirectCFP(Atom,File):-
generateIndirectOLONSet(Atom,File).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% generates the RAA set and asserts it

generateRAAset(File):-
see(File),
loadClauses(raa,null),
generateRAAset,
seen.

generateProgramCopy(File) :-
see(File),
loadClauses(preprocess,null),
seen.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% tails set
generateTailSet([H|T],File):-
generateTails(H,File),
generateTailSet(T,File).

generateTailSet([],_).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for one tail

generateTails(Atom,File) :-
see(File),
loadClauses(tail,Atom),
setof(X,tails(Atom,X),AllTails),
retractall(tails(Atom,_)),
flatten(AllTails,AllTails2),
remove(ni,AllTails2,AllTails3),
assert(tails(Atom,AllTails3)),
seen.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generates a direct cfp for some atom

generateDirectOLONSet(Atom,File):-
see(File),
loadClauses(direct,Atom),
seen.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generates a direct cfp for some atom

generateIndirectOLONSet(Atom,File):-
see(File),
loadClauses(indirect,Atom),
seen.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% loadclauses....

loadClauses(X,Atom) :-
read(C),
( C = end_of_file -> true;
      (
       %write(C),nl,
(
(
X = raa,
processRAA(C)
```

```
);
(
X = direct,
processDirect(C,Atom)
);
(
X = indirect,
processIndirect(C,Atom)
);
(
X = tail,
processTail(Tails,C,Atom),
assert(tails(Atom,Tails))
);
(
X = preprocess,
process(C)
);
true
),
       loadClauses(X,Atom)
      )
    ).


clearall:-
retractall(pos(_)),
retractall(neg(_)),
retractall(raa(_)),
retractall(tails(_,_)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Writes a rule to wherever we're telling/1
%

writeRule(Head) :-
write(Head), write('.'), nl.

writeRule(Head,Tail) :-
write(Head), write(':-'), write(Tail), write('.'), nl.
```

```
translateLiteral(H,NewH,Mode,Atom):-
(
(
Mode = direct,
raa(RAAset),
ith(Iota,RAAset,Atom),
concat_atom([H,Iota,d],NewH)
);
(
Mode = indirect,
raa(RAAset),
ith(Iota,RAAset,Atom),
concat_atom([H,Iota,i],NewH)
)
).


saveTable(Head) :-
write(':-table '),write(Head), write('/0.'), nl.



remove(X,[X|Xs],Ys) :-
!,
remove(X,Xs,Ys).

remove(Z,[X|Xs],[X|Ys]) :-
!,
remove(Z,Xs,Ys).

remove(_,[],[]).
```

B.1.2 raa2.P

```
%%%%%%%%%%%%%%%%%%%%%     A :- B

processRAA((H :- B)) :-
(
```

```
B \= [],
addPos(H),
processRAABody(B)
);
true.


%%%%%%%%%%%%%%%%%%%%%%     A

processRAA((H)) :-
processRAA((H :- [])).


%%%%%%%%%%%%%%%%%%%%%%     Body processing

processRAABody((L,Rest)) :-
addNeg(L),
processRAABody(Rest).


processRAABody(L) :-
addNeg(L).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  we use pos(X) to say that X appeared once in the head of a rule
%%  and neg(X) to say that X appeared once denied in the tail of a rule


addPos(X) :-
retractall(pos(X)),
assert(pos(X)).

addNeg((not X)) :-
retractall(neg(X)),
assert(neg(X)).

addNeg(_).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%
%%  in raa/1 is a list with all the literals which have potential
%%  of being provable by RAA


generateRAAset :-
bagof(X,(pos(X),neg(X)),RAA),
assert(raa(RAA)),
retractall(pos(_)),
retractall(neg(_)).
```

B.1.3 direct.P

```
:- import concat_atom/2 from string.
:- import ith/3 from basics.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%   generation of direct CFPs


processDirect((H :- B),Atom) :-
translateLiteral(H,NewH,direct,Atom),
processDirectBody(B,Atom,NewB),
(
(
raa(RAAset),
ith(_,RAAset,H),
H \= Atom,
%translateLiteral(H,NewH2,direct,H),
concat_atom([H,raa],NewH2),
saveRule(NewH,NewH2)
);
true
),
saveRule(NewH,NewB).

processDirect((H),Atom) :-
```

```
translateLiteral(H,NewH,direct,Atom),
saveFact(NewH).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% body processing

processDirectBody((A ',' R),Atom,NewBody) :-
processDirectBody(A,Atom,NewHead),
processDirectBody(R,Atom,NewTail),
concat_atom([NewHead, ',', NewTail],NewBody).



processDirectBody((not R),Atom,NewAtom):-
(
R \= Atom,
concat_atom([tnot,'(',R,')'],NewAtom),
saveTable(R)
);
(
concat_atom([true],NewAtom)
).


processDirectBody((R),Atom,NewAtom):-
translateLiteral(R,NewAtom,direct,Atom).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    UTILS
%%


translateLiteral(H,NewH,Mode,Atom):-
(
(
Mode = direct,
raa(RAAset),
ith(Iota,RAAset,Atom),
concat_atom([H,Iota,d],NewH)
```

```
);
(
Mode = indirect,
raa(RAAset),
ith(Iota,RAAset,Atom),
concat_atom([H,Iota,i],NewH)
)
).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Writes a rule to wherever we're telling/1
%

saveFact(Head) :-
write(Head), write('.'), nl.

saveRule(Head,Tail) :-
write(Head), write(':-'), write(Tail), write('.'), nl.

saveTable(Head) :-
write(':-table '),write(Head), write('/0.'), nl.
```

B.1.4 indirect.P

```
:- import concat_atom/2 from string.
:- import ith/3 from basics.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%   generation of indirect CFPs


processIndirect((H :- B),Atom) :-
translateLiteral(H,NewH,indirect,Atom),
(
(
raa(RAAset),
ith(_,RAAset,H),
```

```
H \= Atom,
%translateLiteral(H,NewH2,indirect,H),
concat_atom([H,raa],NewH2),
saveRule(NewH,NewH2)
);
true
),
processIndirectBody(B,Atom,NewB),
processIndirectBody2(B,Atom,NewB2),
concat_atom([NewB2,',',tnot,'(',H,')'],NewB3),
saveRule(NewH,NewB),
saveRule(NewH,NewB3).

processIndirect((H),Atom) :-
translateLiteral(H,NewH,indirect,Atom),
saveFact(NewH).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% body processing


processIndirectBody((A ',' R),Atom,NewBody) :-
processIndirectBody(A,Atom,NewHead),
processIndirectBody(R,Atom,NewTail),
concat_atom([NewHead, ',', NewTail],NewBody).

processIndirectBody((not R),Atom,NewAtom):-
(
R \= Atom,
translateLiteral(R,NewR,indirect,Atom),
concat_atom([tnot,'(',NewR,')'],NewAtom),
saveTable(NewR)
);
(
concat_atom([true],NewAtom)
).



processIndirectBody((R),Atom,NewAtom):-
translateLiteral(R,NewAtom,indirect,Atom).
```

```
processIndirectBody2((A ',' R),Atom,NewBody) :-
processIndirectBody2(A,Atom,NewHead),
processIndirectBody2(R,Atom,NewTail),
concat_atom([NewHead, ',', NewTail],NewBody).


processIndirectBody2((not R),Atom,NewAtom):-
(
R \= Atom,
concat_atom([tnot,'(',R,')'],NewAtom)
);
(
concat_atom([true],NewAtom)
).



processIndirectBody2((R),Atom,NewAtom):-
translateLiteral(R,NewAtom,indirect,Atom).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    UTILS
%%


translateLiteral(H,NewH,Mode,Atom):-
(
(
Mode = direct,
raa(RAAset),
ith(Iota,RAAset,Atom),
concat_atom([H,Iota,d],NewH)
);
(
Mode = indirect,
raa(RAAset),
ith(Iota,RAAset,Atom),
concat_atom([H,Iota,i],NewH)
)
).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Writes a rule to wherever we're telling/1
%

saveFact(Head) :-
write(Head), write('.'), nl.

saveRule(Head,Tail) :-
write(Head), write(':-'), write(Tail), write('.'), nl.

saveTable(Head) :-
write(':-table '),write(Head), write('/0.'), nl.
```

B.1.5 tails.P

```
:- dynamic tempTails/1.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%     A :- B

processTail(Tails,(H :- B),Atom) :-
(
assert(tempTails(ni)),
B \= [],
processTailBody(H,B,Atom),
setof(X,tempTails(X),PossibleTails),
Tails = PossibleTails,
retractall(tempTails(_))
);
true.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%     A

processTail(Tails,(H),Atom) :-
processTail(Tails,(H :- []),Atom).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%     Body processing

processTailBody(H,(A ',' R),Atom) :-
processTailBody(H,A,Atom),
processTailBody(H,R,Atom).


processTailBody(H,(not R),Atom):-
R = Atom,
Atom \= H,
assert(tempTails(H)).



processTailBody(_,(R),_).
```

B.1.6 preprocess.P

```
:- import concat_atom/2 from string.



%%%%%%%%%%%%%%%%%%%%%%%%%     A :- B

process((H :- B)) :-
processBody(B,Body),
saveRule(H,Body).



%%%%%%%%%%%%%%%%%%%%%%%%%%%     A

processTail((H)) :-
saveFact(H).


%%%%%%%%%%%%%%%%%%%%%%%%%%%     Body processing

processBody((A ',' R),Body) :-
processBody(A,Body1),
processBody(R,Body2),
```

```
concat_atom([Body1,',',Body2],Body).


processBody((not R),Body):-
concat_atom([tnot,'(',R,')'],Body).



processBody((R),R).



saveRule(Head,Tail) :-
write(Head), write(':-'), write(Tail), write('.'), nl.


saveFact(Head) :-
write(Head), write('.'), nl.
```

B.2 Revised Well-Founded Model calculator

The revised well-founded model calculator simulates the inner working of the Γ^{r^2} operator. As the r part in Γ^{r^2} is actually the transformation performed by the Γ^r operator translator, this implementation actually calculates the well-founded model of a given program, by performing all the steps described in the Γ operator twice.

This implementation is also divided into three modules.

B.2.1 revised.P

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Includes

:- import concat_atom/2 from string.
:- import term_to_atom/2 from string.
:- import term_to_codes/2 from string.
:- import atom_to_term/2 from string.
:- import append/3 from basics.
:- import length/2 from basics.
```

```
:- import member/2 from basics.
:- import flatten/2 from basics.

:- dynamic herbrand/1.



:- consult('gl.P').
:- consult('herbrand.P').

% Main function


calculateRWFM(T,TU,File) :-
getT(T,File,[]),
gamma(T,TU,File).

getT(T,File,I) :-
gammaSquared(I,M,File),
I = M,!,
T = I.

getT(T,File,I) :-
gammaSquared(I,M,File),
getT(T,File,M).


gammaSquared(I,M,File) :-
gamma(I,M1,File),
gamma(M1,M,File),!.

gammaSquared(I,M,File).




gamma(I,M,File) :-
clearall,
tell('p_modulo_i.P'),
glTransform(I,File),
told,
leastModel,
herbrand(H),
callTp(H,M),!.
```

```
gamma(I,M,File).

glTransform(I,File) :-
see(File),
loadClauses(I,ni),
seen.


leastModel:-
see('p_modulo_i.P'),
loadClauses([],herbrand),
seen,
load_dyn('p_modulo_i.P'),
setof(X,herbrand(X),AllHerbrands),
retractall(herbrand(_)),
flatten(AllHerbrands,AllHerbrands2),
remove(true,AllHerbrands2,AllHerbrands3),
remove(fail,AllHerbrands3,AllHerbrands4),
removeDuplicates(AllHerbrands4,H),
assert(herbrand(H)).
%write(H),nl.
%callTp(H,M),
%write('sai...'),nl.



callTp([H|T],[H|R]) :-
call(H),
callTp(T,R).

callTp([H|T],R) :-
not call(H),
callTp(T,R).

callTp([],[]).



loadClauses(I,Mode) :-
read(C),
( C = end_of_file -> true;
      (
```

```
       %write(C),nl,
(
Mode = herbrand,
processHerbrand(C,MiniHerbrand),
assert(herbrand(MiniHerbrand))
);
(
processGL(C,I)
);
true
  ),
loadClauses(I,Mode)
    ).



clearall:-
retractall(herbrand(_)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Writes a rule to wherever we're telling/1
%

writeRule(Head) :-
write(Head), write('.'), nl.

writeRule(Head,Tail) :-
write(Head), write(':-'), write(Tail), write('.'), nl.




memberList(X,[X|_]).
memberList(X,[_|Tail]):-
   memberList(X,Tail).

removeDuplicates([],[]).

removeDuplicates([X|L],Pruned):-
memberList(Y,L), X==Y, !,
removeDuplicates(L,Pruned).
```

```
removeDuplicates([X|L],[X|Pruned]):-
removeDuplicates(L,Pruned).
```

B.2.2 herbrand.P

```
:- import concat_atom/2 from string.
:- import ith/3 from basics.
:- import member/2 from basics.


:- dynamic tempHerbrand/1.


processHerbrand((H :- B),MiniHerbrand) :-
H \= [],
assert(tempHerbrand(H)),
processHerbrandBody(B),
setof(X,tempHerbrand(X),PossibleHerbrand),
MiniHerbrand = PossibleHerbrand,
retractall(tempHerbrand(_)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% body processing


processHerbrandBody((A ',' R)) :-
processHerbrandBody(A),
processHerbrandBody(R).


processHerbrandBody((R)) :-
assert(tempHerbrand(R)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Writes a rule to wherever we're telling/1
%

saveFact(Head) :-
write(Head), write('.'), nl.
```

```
saveRule(Head,Tail) :-
write(Head), write(':-'), write(Tail), write('.'), nl.

saveTable(Head) :-
write(':-table '),write(Head), write('/0.'), nl.
```

B.2.3 gl.P

```
:- import concat_atom/2 from string.
:- import ith/3 from basics.
:- import member/2 from basics.


processGL((H :- B),I) :-
H \= [],
processGLBody(B,NewBody,I),
saveRule(H,NewBody),
saveDynamic(H).

processGL((H),_) :-
saveRule(H,true).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% body processing


processGLBody((A ',' R),NewBody,I) :-
processGLBody(A,NewHead,I),
processGLBody(R,NewTail,I),
concat_atom([NewHead, ',', NewTail],NewBody).




processGLBody((tnot(R)),NewAtom,I):-
(
member(R,I),
concat_atom([fail],NewAtom)
);
concat_atom([true],NewAtom).
```

```
processGLBody((R),R,_):-
saveDynamic(R).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Writes a rule to wherever we're telling/1
%

saveFact(Head) :-
write(Head), write('.'), nl.

saveRule(Head,Tail) :-
write(Head), write(':-'), write(Tail), write('.'), nl.

saveDynamic(Head) :-
write(':-dynamic '),write(Head), write('/0.'), nl.
```