Reductio ad Absurdum Argumentation in Normal Logic Programs

Luís Moniz Pereira and Alexandre Miguel Pinto
{lmp|amp}@di.fct.unl.pt

Centro de Inteligência Artificial (CENTRIA)
Universidade Nova de Lisboa
Quinta da Torre
2829-516 Caparica, Portugal

Abstract. This paper introduces a new method for defining the argumentative semantics of Normal Logic Programs. In doing so, our single and unified approach allows one to obtain the Stable Models [11] as a special case, or the more general Revision Complete Scenarios here defined.
Normal Logic Programs are approached as assumption-based argumentation systems. We generalize this setting by allowing both negative and positive assumptions. Negative assumptions are made maximal, consistent with existence of a semantics, and positive assumptions are adopted only insofar as they guarantee such existence. Our argumentation semantics thus extends the classical one of [7], and guarantees existence of semantics for any Normal Logic Program, whilst providing all the scenarios corresponding to Stable Models semantics.
Additionally, we provide equivalent and correct algorithms for incrementally computing our scenarios, with three variants. One starts by assuming all atoms as positive assumptions; another assumes them all negative; a third rests on a combination of the first two, and may start with any choice of assumptions. The latter may be employed to address the problem of finding those complete scenarios most compatible with an initial collection of complete scenarios. Consequently, argumentation can be put to collaborative use, not just an antagonistic one. Our results are achieved by generalizing the definitions of the classical approach, which allows only for negative hypotheses, and our definitions fall back on the classical ones when specialized to disallow positive hypotheses.
Finally, integrity constraints are introduced to prune undesired scenarios, whilst permitting these to be produced nevertheless.
Keywords: Argumentation, *Reductio ad Absurdum*, Logic Programs, Argument Revision

1 Introduction

After introducing in [15] and [14] the new Revised Stable Models semantics for Normal Logic Programs further work using the *Reductio ad Absurdum* (RAA) principle has been developed, namely the Revised Well-Founded Semantics [16]. Considering an argument-based view of Logic Programs, we define a new semantics which inherits the RAA principle studied in [15, 14] and apply it to argumentation.

Logic Programs can be viewed as a collection of argumentative statements (rules) based on arguments (default negated literals) [5, 2, 6, 17, 3, 13, 9, 8, 7]. In the quest for finding a Consistent and Complete argumentative scenario one can guess it and check its compliance with these properties; or, innovatively, start with an arbitrary scenario, calculate its consequences, and make revisions to the initial assumptions if necessary in order to achieve 2-valued Completeness and Consistency. This is the road we propose now, revision of assumptions justified by means of *Reductio ad Absurdum* reasoning.

This paper introduces a new method for defining the argumentative semantics of Normal Logic Programs. In doing so, our single and unified approach allows one to get the Stable Models [11] as a special case, or the more general Revision Complete Scenarios here defined.

Normal Logic Programs are approached as assumption-based argumentation systems. We generalize this setting by allowing both negative and positive assumptions. Negative assumptions are made maximal, consistent with existence of a semantics, and positive assumptions are adopted only insofar as they guarantee such existence. The justification of positive assumptions rests on the use of *reductio ad absurdum*, to the effect that replacing any one positive hypothesis (or assumption) by its negative counterpart, in a complete scenario, would result in its inconsistency. Hence, that complete 2-valued scenario must retain its positive assumptions. Our argumentation semantics thus extends the classical one of [7], and guarantees existence of semantics for any Normal Logic Program, whilst providing all the scenarios corresponding to Stable Models semantics.

Additionally, we provide equivalent and correct algorithms for incrementally computing our scenarios, with three variants. One starts by assuming all atoms as positive assumptions; another assumes them all negative; a third rests on a combination of the first two, and may start with any choice of assumptions. The latter may be employed to address the problem of finding those complete scenarios most compatible with an initial collection of complete scenarios. Consequently, argumentation can be put to collaborative use, not just an antagonistic one. Our results are achieved by generalizing the definitions of the classical approach, which allow only for negative hypotheses, and our definitions fall back on the classical ones when specialized to disallow positive hypotheses.

Finally, integrity constraints are introduced to prune undesired scenarios, whilst permitting these to be produced nevertheless.

In essence, our approach caters for the treatment of loops over an odd number of default negated literals, in that it assigns and justifies complete 2-valued models to any Normal Logic Program.

We start by presenting the general Motivation of this paper and, after introducing some needed Background Notation and Definitions, the more detailed Problem Description. We proceed by setting forth our proposal — the Revision Complete Scenarios— and show how it extends previous known results.

Before the Conclusions and Future Work, we show how our approach can enable Collaborative Argumentation, complementing the classical Competitive view of Argumentation.

1.1 Motivation

Ever since the beginning of Logic Programming the scientific community has formally define, in several ways, the meaning, the semantics of a Logic Program. Several semantics were defined, some 2-valued, some 3-valued, and even multi-valued semantics. The current standard 2-valued semantics for Normal Logic Programs— the Stable Models Semantics [11] — has been around for almost 20 years now, and it is generally accepted as the *de facto* standard 2-valued semantics for NLPs. This thoroughly studied semantics, however, lacks some important properties among which the guarantee of Existence of a Model for every NLP.

In [14] we defined a 2-valued semantics— the Revised Stable Models— which extends the Stable Models Semantics, guarantees Existence of a Model for every Normal Logic Program, enjoys Relevancy (allowing for top-down query-driven proof-procedures to be built) and Cumulativity (allowing the programmer to take advantage of tabling techniques for speeding up computations).

Aiming to find a general perspective to seamlessly unify the Stable Models Semantics and the Revised Stable Models Semantics we drew our attention to Argumentation as a means to achieve it. This is the main motivation of the work we present in this paper: by taking the Argumentation perspective we intend to show methods of identifying and finding a 2-valued complete Model for any NLP. The approach is unifying in the sense that it allows us to find the Stable Models and also some other Models needed to ensure guarantee of Existence of a Model. In the process we extend the argumentation stance itself with the ability to incorporate positive hypotheses as needed.

Example 1. **An invasion problem** Some political leader thinks that "If Iran will have Weapons of Mass Destruction then we intend to invade Iran", also "If we do not intend to invade then surely they will have Weapons of Mass Destruction".

$$\begin{aligned} intend\_we\_to\_invade &\leftarrow iran\_will\_have\_WMD \\ iran\_will\_have\_WMD &\leftarrow not\ intend\_we\_to\_invade \end{aligned}$$

If we assume that "we do not intend to invade Iran" then, according to this program we will conclude that "Iran will have Weapons of Mass Destruction" and "we intend to invade Iran". These conclusions, in particular "we intend to invade Iran", contradict the initial hypothesis "we do not intend to invade Iran". So, reasoning by *Reductio ad Absurdum* in a 2-valued setting, we should "intend to invade Iran" in the first place.

This example gives a hint on how we resolve inconsistent scenarios in the rest of the paper.

Example 2. **A vacation problem** Another example puts together three friends that are discussing where they will spend their next joint vacations. John says "If I cannot go the mountains I'd rather go traveling". Mary says "Well, I want to go to the beach, but if that's not possible then I'd rather go to the mountains". Finally, Michael says "I want to go traveling, and if that's not possible then I want to go to the beach".

We put together the three friends' statements formalized into a Normal Logic Program:

$$travel \leftarrow not\ mountain \quad mountain \leftarrow not\ beach \quad beach \leftarrow not\ travel$$

Now, because the three friends need to save money, they must minimize the number of places they will go to on vacation. So they start by assuming they are going nowhere — the cheapest solution. That is, they assume $\{not\ mountain, not\ beach, not\ travel\}$ as true. According to the program above, with these initial hypotheses the friends will conclude they will go traveling, to the beach and to the mountains; and this contradicts the initial hypotheses. They need to revise some of their initial assumptions. If they revise $not\ mountain$ to $mountain$ they will now conclude $\{mountain, beach\}$ and if we put it together with the new set of hypotheses $\{not\ beach, not\ travel, mountain\}$ we get the resulting set $\{mountain, beach, not\ beach, not\ travel\}$. We still have a contradiction on $beach$ and $not\ beach$, which we can easily remove by transforming the hypotheses set into $\{mountain, beach, not\ travel\}$.

There are two more alternative solutions — $\{beach, travel, not\ mountain\}$ and $\{travel, mountain, not\ beach\}$ — which are symmetric to this one.

Example 3. **A time-out problem** John likes Mary a lot so he asked her out: he said "We could go to the movies". Mary is more of a sports girl, so she replies "Either that, or we could go to the swimming pool". "Now, that's an interesting idea", John thought. The problem is that John cannot swim because he hasn't started learning to. He now thinks "Well, if I'm going to the swimming pool with Mary, and I haven't learned how to swim, I'm might risk drowning! And if I'm risking drowning then I really should want to start learning to swim".

Here is the Normal Logic Program corresponding to these sentences:

$$\begin{array}{ll} start\_learning\_to\_swim & \leftarrow risk\_drowning \\ risk\_drowning & \leftarrow go\_to\_pool, not\ start\_learning\_to\_swim \\ go\_to\_pool & \leftarrow not\ go\_to\_movies \\ go\_to\_movies & \leftarrow not\ go\_to\_pool \end{array}$$

If John is not willing to go to the swimming pool — assuming $not\ go\_to\_pool$ — he just concludes go_to_movies and maybe he can convince Mary to join him.

On the other hand, if the possibility of having a nice swim with Mary is more tempting, John assumes he is not going to the movies $not\ go\_to\_movies$ and therefore he concludes go_to_pool. In this case, since John does not know how to swim he could also assume $not\ start\_learning\_to\_swim$. But since John is going to the swimming pool, he concludes $risk\_drowning$. And because of $risk\_drowning$ he also concludes $start\_learning\_to\_swim$. That is, he must give up the hypothesis of $not\ start\_learning\_to\_swim$ in favor of $start\_learning\_to\_swim$ because he wants to go to the swimming pool with Mary. As a nice side-effect he no longer risks drowning.

Example 4. **Middle Region Politics** In a Middle Region two factions are at odds. One believes that if terrorism does not stop then oppression will do it and hence become

unnecessary.

$$oppression \leftarrow not\ end\_of\_terrorism \quad end\_of\_terrorism \leftarrow oppression$$

The other faction believes that if oppression does not stop then terrorism will do it and hence become unnecessary.

$$terrorism \leftarrow not\ end\_of\_oppression \quad end\_of\_oppression \leftarrow terrorism$$

According to these rules, if we assume the $not\ end\_of\_terrorism$ we conclude that there is $oppression$ which in turn will cause the $end\_of\_terrorism$. So, the $end\_of\_terrorism$ should be true in the first place, instead of $not\ end\_of\_terrorism$. The same happens with $end\_of\_oppression$. In spite of the peaceful resulting scenario we propose, $\{end\_of\_oppression, end\_of\_terrorism\}$, there is no Stable Model for this program.

1.2 Background Notation and Definitions

Definition 1. ***Logic Rule*** *A Logic Rule r has the general form $L \leftarrow b_1, b_2, \ldots, b_n, not\ c_1, not\ c_2, \ldots, not\ c_m$ where L is a literal, i.e., an atom h or its default negation $not\ h$, and $n, m \geq 0$.*

We call L the head of the rule — also denoted by $head(r)$. And $body(r)$ denotes the set $\{b_1, b_2, \ldots, b_n, not\ c_1, not\ c_2, \ldots, not\ c_m\}$ of all the literals in the body of r. Throughout this paper we will use 'not ' to denote the default negation.

When the body of the rule is empty, we say the head of rule is a fact and we write the rule as just h or $not\ h$. □

Definition 2. ***Logic Program*** *A Logic Program (LP for short) P is a (possibly infinite) set of ground Logic Rules of the form presented in definition 1. If the heads of all the rules in P are positive literals, i.e., they are simple atoms, and not default negated literal, we say we have a Normal Logic Program (NLP). If at least one of the heads of a rule of P is a default negated literal, and there is no explicit negation in the program — we say we have a Generalized Logic Program (GLP). If there is explicit negation, besides default negation, in the program we say we have an Extended Logic Program (ELP).* □

Definition 3. ***Atoms of a Logic Program*** *P — $Atoms(P)$ $Atoms(P)$ denotes the set of all atoms of P. Formally,*

$Atoms(P) = \{a : \exists_{r \in P}\big(head(r) = a \vee head(r) = not\ a \vee a \in body(r) \vee not\ a \in body(r)\big)\}$ □

Throughout the rest of this paper we will focus solely on Normal Logic Programs hence, when we write just a Program or a Logic Program we mean a Normal Logic Program.

Definition 4. ***Default negation of a set S of literals —*** *$not\ S$ Throughout this paper we will sometimes use the $not\ S$ default negation of a set S notation, where S is a set of literals, in order to denote the set resulting from default negating every literal of S. Formally, $not\ S = \{not\ a : a \in S\} \cup \{b : not\ b \in S\}$* □

Definition 5. ***Scenario*** *A scenario of a NLP P is the Horn theory $P \cup H$, where $H = H^+ \cup H^-$, $H^+ \subseteq Atoms(P)$, $H^- \subseteq not\ Atoms(P)$, and $not\ H^+$ and H^- are disjoint. H is called a set of hypotheses, positive and negative.* □

Definition 6. $\vdash$ ***operator*** *Let P be a NLP and H a set of hypotheses. P' is the Horn theory obtained from P by replacing every default literal of the form $not\ L$ in P by the atom not_L. H' is likewise obtained from H using the same replacement rule. By definition, $P' \cup H'$ is a Horn theory, and so it has a least model M. We define $\vdash$ in the following way, where A is any atom of P:*

$P \cup H \vdash A$ *iff* $A \in M$ $\qquad$ $P \cup H \vdash not\ A$ *iff* $not\_A \in M$ □

Definition 7. ***Consistent scenario*** *A scenario $P \cup H$ is consistent iff for all literals L, if $P \cup H \vdash L$ then $P \cup \vdash H \nvdash not\ L$, where $not\ not\ L \equiv L$.* □

Definition 8. ***Consistent program*** *A Logic Program P is consistent iff $P \cup \emptyset$ is a consistent scenario. NLPs are of course consistent.* □

2 Revision Complete Scenarios

In [4] the author proves that every Stable Model (SM) of a NLP is a 2-valued complete (total), consistent, admissible scenario. The author considers a scenario as a set of default negated literals — the hypotheses. However, not every NLP has a consistent, 2-valued complete scenario when one considers as hypotheses just default negated literals.

Also in [4], the author shows that preferred maximal (with maximum default negated literals) scenarios are always guaranteed to exist for NLPs. However, preferred maximal scenarios are, in general, 3-valued.

The problem we address now is to find a way to render 2-valued total a preferred maximal scenario. In this paper we take a step further from what was previously achieved in [4], extending its results. We allow a set of hypotheses to contain also positive literals, but only those absolutely necessary to guarantee Existence of a Model. These positive hypotheses are those who are justified *true* by a specific *Reductio ad Absurdum* reasoning we accept.

Before presenting the formal Definition of a Revision Complete Scenario we give a general intuitive idea to help the reader grasp the concept. For the formal definition of Revision Complete Scenario we will also need some preliminary auxiliary definitions.

2.1 Intuition

In [3] the authors prove that every SM of a NLP corresponds to a stable set of hypotheses which correspond in turn to a 2-valued complete, consistent, admissible scenario.

In order to guarantee the Existence of a 2-valued total Model for every NLP we allow positive hypotheses to be considered besides the usual negative hypotheses. Under this setting, the easiest way to solve the problem would be to accept every atom of a program as a positive hypotheses. However, we want to our semantics to be the most skeptical possible while ensuring stratification compatibility among hypotheses.

To further keep the semantics skeptical we want to have the maximal possible negative hypotheses and the minimum non-redundant positive hypotheses. Intuitively, a positive hypothesis L is considered redundant if, by the rules of the program and the rest of the hypotheses, L is already determined *true*. The formal definition of this notion of non-redundancy of positive hypotheses is presented and explained below.

The formal notion of compatibility will also be depicted and explained below, but for now the intuitive idea is that one positive hypothesis L must not contradict other hypotheses.

2.2 Definition

Definition 9. ***Evidence for a literal*** *L A negative set of hypotheses $E \subseteq not\ Atoms(P)$ is evidence for a literal L in program P iff $P \cup E \vdash L$. If P is understood we write $E \rightsquigarrow L$. We also say E attacks $not\ L$. Notice that we do not require an evidence to be consistent.* □

Definition 10. ***Weakly Admissible set of hypotheses*** H^-

The notion of weakly admissible set presented here is in line with that of weak stability, first defined in [12].

Let P be a NLP, $H^- \subseteq not\ Atoms(P)$ a set of negative hypotheses, $not\ L$ a default negated literal in P and E an evidence for L. We say H^- is weakly admissible iff $\forall_{not\ L \in H^-} \forall_{E \rightsquigarrow L} \exists_{not\ A \in E} P \cup H^- \cup E \vdash A$ □

The classical notion of admissible set checks only if $P \cup H^- \vdash A$. By doing this test with $P \cup H^- \cup E$ we allow E to be inconsistent. It suffices to see that if $P \cup H^- \nvdash A$ and $P \cup H^- \cup E \vdash A$ it means that E is essential to derive A in the $P \cup H^-$ context. Since we know $not\ A \in E$ and $P \cup H^- \cup E \vdash A$ we conclude that E is inconsistent.

There are some sets of hypotheses H^- which were not admissible according to the classical definition (with just $P \cup H^-$) and are weakly admissible — according to the definition using $P \cup H^- \cup E$. These sets of hypotheses which are accepted as weakly admissible are just the ones where the adding of the evidence E was essential to derive A, that is, where E is inconsistent.

Since the $\vdash$ operator is monotonic, every admissible set of hypotheses according to the classical definition (using $P \cup H^-$) is also weakly admissible — according to the definition with $P \cup H^- \cup E$.

Example 5. **Weakly Admissible vs Non Weakly Admissible sets of negative hypotheses** Consider the following NLP:

$$k \leftarrow not\ t \quad t \leftarrow a, b \quad a \leftarrow not\ b \quad b \leftarrow not\ a$$

In this program we can easily see that the bottom Even Loop Over Negation (ELON, for short) over a and b allows only one of them to be true — when we demand minimality of positive information. Under this setting we will never have t true for it needs both a and b to be true simultaneously to support its truthfulness. Therefore, k will always be true, since t is always false.

Let us analyze the different possible sets of hypotheses from an admissibility point of view. Consider the following two sets of negative hypotheses $H_1 = \{not\ b, not\ t\}$ and $H_2 = \{not\ b, not\ k\}$. The other two sets of negative hypotheses H_3 and H_4 are just symmetric to H_1 and H_2, respectively, on $not\ a$ and $not\ b$; therefore we are going to focus solely on H_1 and H_2.

H_1 is weakly admissible whereas H_2 is not. Let us see why. Analyzing $not\ b$ we verify that there is only one possible evidence $E = \{not\ a\}$ for b and that $P \cup H_1 \cup E \vdash a$, i.e., $H_1 \cup E$ attacks (in the sense presented in definition 9) $not\ a$. In this particular case even just H_1 attacks $not\ a$.

Analyzing $not\ t$ we can see that there is only one evidence $E = \{not\ a, not\ b\}$ for t. $P \cup H_1 \cup E$ derives both a and b, i.e., $P \cup H_1 \cup E \vdash a$ and $P \cup H_1 \cup E \vdash b$; hence H_1 is weakly admissible.

Let us see what happens with H_2. We have already seen $not\ b$, we just need to test $not\ k$. The only evidence for k is $E = \{not\ t\}$. We can see however that $P \cup H_2 \cup E \nvdash t$, which leads us to conclude that H_2 is not weakly admissible.

Example 6. **Allowing Inconsistent Evidence** Consider the following NLP:

$$k \leftarrow not\ t \quad t \leftarrow not\ t$$

The hypotheses $H_1 = \{not\ t\}$ is admissible and weakly admissible. However, since $P \cup H_1$ is not a consistent scenario, no model exists with $not\ t$.

The only possible hypotheses left are the empty set and $H_2 = \{not\ k\}$. Considering the classical notion of admissible set (with $P \cup H^-$) H_2 is non-admissible; however, H_2 is weakly admissible. Notice that the evidence for k is $E = \{not\ t\}$ and that $P \cup H_2 \cup E \vdash t$. $P \cup H_2$ is a consistent scenario, but it is not complete. Since we already know that $not\ t$ cannot be in any consistent model, in a 2-valued setting we would like to "complete" the scenario $P \cup H_2$ with t in order to obtain a 2-valued complete and consistent model. In such case we say $\{t\}$ is our set of positive hypotheses.

Definition 11. ***Non-redundant set H^+ of positive hypotheses*** *Let P be a NLP, and $H = H^+ \cup H^-$ a set of positive and negative hypotheses, i.e., ($H^+ \subseteq Atoms(P)$) and ($H^- \subseteq not\ Atoms(P)$). We say H^+ is non-redundant iff* $\forall_{L \in H^+} P \cup H \setminus \{L\} \nvdash L$ □

As just explained, we wish to allow some positive hypotheses when they are absolutely needed in order to obtain 2-valued complete and consistent scenarios. However, we require the positive set of hypotheses to be non-redundant, that is, all positive hypotheses must not be already derived by other hypotheses. This is the purpose of definition 11 above.

Example 7. **Redundant positive hypotheses** Consider the following program P:

$$b \leftarrow a \quad a \leftarrow not\ a$$

In the previous example 6 we saw how a rule like $t \leftarrow not\ t$ forbids the negative hypothesis $not\ t$. By the same token, in this example's program, the hypothesis $not\ a$ is also forbidden. Also $\{not\ b\}$ is not a weakly admissible set of negative hypotheses.

Since we are looking for 2-valued complete (total) and consistent scenarios, we would like one including both a and b.

The question now is: should both a and b be considered positive hypotheses? Since we are looking for the minimum possible set of positive hypotheses (compatible with the negative ones), we answer *no* in this case, because assuming the positive hypothesis a is enough to automatically determine the truth of b. That is why we say the set $\{a, b\}$ of positive hypotheses is redundant, whereas $\{a\}$ is not.

Definition 12. ***Unavoidable set H^+ of positive hypotheses*** *Let P be a NLP, and $H = H^+ \cup H^-$ a set of positive and negative hypotheses. We say H^+ is unavoidable iff $\forall_{L \in H^+} P \cup \big(H \setminus \{L\}\big) \cup \{not\ L\}$ is an inconsistent scenario* □

In a nutshell, this definition imposes that every positive hypothesis must be accepted as true for the sake of consistency and completeness in the context of all other hypotheses. We ensure this by demanding that any if positive hypothesis L was to be considered false — i.e., $not\ L$ considered true — the whole scenario of P with all the hypotheses, except L, and including $not\ L$ instead (for the sake of 2-valued completeness) would be inconsistent. So, there is no consistent 2-valued way to avoid having L true in the context of the remaining hypotheses. Additionally, one may need the condition as stating that, if the scenario with $not\ L$ is consistent, then L is avoidable.

Example 8. **Unavoidable vs Avoidable sets of positive hypotheses** Let P be the following NLP:

$$d \leftarrow not\ c \quad c \leftarrow not\ b \quad b \leftarrow not\ a \quad a \leftarrow not\ a$$

In this example we consider $H_1 = H_1^+ \cup H_1^-$, where $H_1^+ = \{a\}$ and $H_1^- = \{not\ b, not\ d\}$; and $H_2 = H_2^+ \cup H_2^-$, where $H_2^+ = \{a, b\}$ and $H_2^- = \{not\ c\}$.

By the same reason as in example 7 $not\ a$ cannot be in any H^- and, in order to obtain a 2-valued total model with an H, a must be accepted as true — in that sense we say a is unavoidable.

Definition 13. ***Revision Complete Scenarios*** *Let P be a NLP and $H = H^+ \cup H^-$ a set of positive (H^+) and negative (H^-) hypotheses. We say H is a Revision Complete Scenario iff*

1. *$P \cup H$ is a consistent scenario and $least(P \cup H)$ is a 2-valued complete model of P*
2. *H^- is weakly admissible*
3. *H^+ is not redundant*
4. *H^+ is unavoidable*

□

2.3 The Exhaustive Model Generation Algorithm

Another method for finding the Revision Complete Scenarios is an iterative and incremental way.

Definition 14. ***Inconsistency avoidance algorithm for generating the Revision Complete Scenarios (RCSs)***

1. *Start with* $i = 0$, $H_i^+ = Atoms(P)$ *and* $H_i^- = \emptyset$.
2. *If* H_i^- *is not weakly admissible then* $H_i^+ \cup H_i^-$ *is not a Revision Complete Scenario and the algorithm terminates unsuccessfully.*
3. *If* H_i^- *is weakly admissible then:*
4. *If* $H_i^+ = \emptyset$ *then* $H_i^+ \cup H_i^-$ *is a RCS and the algorithm terminates successfully in this case.*
5. *If* $H_i^+ \neq \emptyset$ *then non-deterministically take one arbitrary* $L \in H_i^+$ *and check if* H_i^+ *is redundant on* L. *If it is then:*
6. $H_{i+1}^+ = H_i^+ \setminus \{L\}$ *and go back to step 3 (a).*
7. *If* H_i^+ *is non-redundant then:*
8. *Check if* H_i^+ *is unavoidable and, if so, then* $H_i^+ \cup H_i^-$ *is a RCS and the algorithm terminates successfully.*
9. *If* H_i^+ *is not unavoidable and* $L \in H_i^+$ *is one of the positive hypotheses rendering* H_i^+ *non-unavoidable then* $H_{i+1}^+ = H_i^+ \setminus \{L\}$ *and* $H_{i+1}^- = H_i^- \cup \{not\ L\}$ *and go on to step 2 again.*

□

This algorithm starts with all the possible positive hypotheses (all the atoms of the program) and no negative hypotheses. By construction, a scenario with such H^+ and H^- is necessarily consistent and 2-valued complete. Along the execution of the algorithm, at each time, we either just remove one positive hypothesis because redundant, or non-deterministically remove one positive hypothesis and add its correspondent default negation to the set of negative hypotheses. By construction, the algorithm guarantees that $H = H^+ \cup H^-$ is consistent. When we just remove one positive hypothesis $L \in H^+$ the 2-valued completeness of the resulting scenario is guaranteed because L was removed from H^+ only because L was rendering H^+ redundant. When we remove L from H^+ and add $not\ L$ to H^- 2-valued completeness is naturally assured.

The requirement for weak admissibility of H^- in step 3 ensures the resulting $H = H^+ \cup H^-$ corresponds to a consistent scenario. The different non-deterministic choices engender all the RCSs.

Example 9. **Generating RCSs by Inconsistency avoidance**

$$a \leftarrow not\ a, not\ b \qquad b \leftarrow not\ a, not\ b$$

We start the algorithm with all the possible positive hypotheses and no negative ones:

- $H_0^+ = \{a, b\}, H_0^- = \emptyset$.
- H_0^- is weakly admissible.
- $H_0^+ \neq \emptyset$ so we check if it is redundant. It is not, so we check if H_0^+ is unavoidable.
- H_0^+ is not unavoidable. We non-deterministically choose one atom from $H_0^+ = \{a, b\}$ which makes it non-unavoidable (in this case, both a and b are rendering H_0^+ non-unavoidable, so we can choose any one). Let us say we choose b. Then $H_1^+ = H_0^+ \setminus \{b\}$ and $H_1^- = H_0^- \cup \{not\ b\}$. And we go on to step 2 again.

- H_1^- is weakly admissible.
- $H_1^+ \neq \emptyset$.
- H_1^+ is not redundant on any $L \in H_1^+$.
- H_1^+ is unavoidable and so $H_1 = H_1^+ 1 \cup H_1^- = \{a, not\ b\}$ is a Revision Complete Scenario and the algorithm terminates successfully.

If we were to choose $not\ a$ instead of $not\ b$ in step 9, the resulting Revision Complete Scenario would be $\{not\ a, b\}$. There are no other Revision Complete Scenario for this program besides these two.

Theorem 1. ***The sets*** $H = H^+ \cup H^-$ ***resulting from the execution of algorithm of definition 14 are the Revision Complete Scenarios***

Proof. Trivial, by construction of the algorithm. □

Theorem 2. ***Existence of Model*** *For any given NLP P there is always at least one Revision Complete Scenario.*

Proof. In the algorithm described above, when we need to non-deterministically choose one atom L to remove from H_i^+, and eventually add $not\ L$ to H_i^-, if there are no repetitions in the choice, then the algorithm is necessarily guaranteed to terminate.

Moreover, if the first positive hypothesis to remove correspond to atoms upon which no other atoms depend, then removing that positive hypotheses has causes no inconsistency, nor does it compromise 2-valued completeness. If the next positive hypotheses in the sequence to be removed always guarantee that the consequences of its removal (and eventual adding of its default negated counterpart to the set of negative hypotheses) does not change the truth value of positive hypotheses already removed, then it is necessarily guaranteed that the algorithm will find a Revision Complete Scenario.

Finally, it is always possible to find such a sequence of positive hypotheses to remove: the sequence just needs to be in reverse order of the stratification of the program. I.e., the first positive hypotheses in the sequence must be from the top strata of the program, the second hypotheses from the second strata counting from the top, and so on. The notion of stratification we are unsing here can be intuitively explained as: (1) atoms in a loop are all in the same strata; (2) atoms which are not in a loop, and are in the head of a rule are in a strata which is always one directly above the atoms in the body of the rule. □

Theorem 3. M ***is a Stable Model of a NLP*** P ***iff there is some Revision Complete Scenario*** H ***such that*** $M = least(P \cup H)$ ***with*** $H^+ = \emptyset$

Proof. Let $H = H^+ \cup H^-$ a set of positive and negative hypotheses. Let us consider the particular case where $H^+ = \emptyset$, therefore $H = H^-$.

In [4], the author already proved that when $H = H^-$, $P \cup H$ is a consistent scenario and $M = least(P \cup H)$ is a 2-valued complete scenario iff M is a Stable Model of P.

Stable Models are just a particular case of Revision Complete Scenarios. □

A variation of this algorithm reversing the direction of the changes in H^+ and H^- can also be depicted. In such an algorithm we start with $H^- = not\ Atoms(P)$ and $H^+ = \emptyset$. 2-valued completeness is also assured at the starting point, although consistency of $P \cup H$ is not. The algorithm is:

Definition 15. ***Inconsistency removal algorithm for generating the Revision Complete Scenarios (RCSs)***

1. *Start with* $i = 0$, $H_i^- = not\ Atoms(P)$ *and* $H_i^+ = \emptyset$.
2. *If* $P \cup H_i$ *is a consistent scenario then* H_i *is a RCS and the algorithm terminates successfully.*
3. *Check if* H_i^+ *is redundant:*
4. *If it is redundant then non-deterministically take one arbitrary atom* $L \in H_i^+$ *such that* $P \cup H \setminus \{L\} \vdash L$ *and construct* $H_{i+1}^+ = H_i^+ \setminus \{L\}$.
5. *If* H_i^+ *is non-redundant construct* $H_{i+1}^+ = H_i^+$.
6. *Check if* H_{i+1}^+ *is unavoidable:*
7. *If* H_{i+1}^+ *is non-unavoidable then* $H_{i+1}^+ \cup H_{i+1}^-$ *is not a RCS and the algorithm terminates unsuccessfully.*
8. *If* H_{i+1}^+ *is unavoidable then check if* $P \cup H_{i+1}$ *is a consistent scenario:*
9. *If* $P \cup H_{i+1}$ *is a consistent scenario then:*
10. *Check if* $P \cup H_{i+1}$ *is also a 2-valued complete scenario and if it is then* H_{i+1} *is a RCS and the algorithm terminates successfully.*
11. *If* $P \cup H_{i+1}$ *is not a 2-valued complete scenario then construct* $H_{i+2}^+ = H_{i+1}^+ \cup \{L\}$, *where* $P \cup H_{i+1} \nvdash L$ *and* $P \cup H_{i+1} \nvdash not\ L$, *and* H_{i+2}^+ *is non-redundant. Go on to step 4 again.*
12. *If* $P \cup H_{i+1}$ *is not a consistent scenario, take one* $not\ L \in H_{i+1}^-$ *such that* $P \cup H_{i+1} \vdash L$ *and* $P \cup H_{i+1} \vdash not\ L$ *(i.e., there is a contradiction in* L *with* $P \cup H_{i+1}^-$*) and construct* $H_{i+2}^- = H_{i+1}^- \setminus \{not\ L\}$ *and* $H_{i+2}^+ = H_{i+1}^+ \cup \{L\}$, *i.e., we revise the assumption* $not\ L$ *to* L *making it a positive hypothesis. Go on to step 3 again.*

□

This algorithm starts with all the possible negative hypotheses (the default negation of all the atoms of the program) and no positive hypotheses. By construction, a scenario with such H^+ and H^- is necessarily consistent and 2-valued complete. Along the execution of the algorithm, at each time, we either just remove one positive hypothesis — because it is redundant — , or remove one negative hypothesis $not\ L$ and add its correspondent positive L to the set of positive hypotheses — i.e., we revise the assumption $not\ L$ to L, when the set of negative hypotheses with $not\ L$ is not consistent.

Also by construction the algorithm guarantees that $H = H^+ \cup H^-$ is consistent and, therefore, that H^- is weakly admissible. When we just remove one positive hypothesis $L \in H^+$ the 2-valued completeness of the resulting scenario is guaranteed because L was redundant in H^+. When we remove $not\ L$ from H^- and add L to H^+ 2-valued completeness is naturally assured. The different non-deterministic choices engender all the RCSs.

Example 10. **Generating RCSs by Inconsistency removal** Let us revisit the example 9 and see the Inconsistency removal version of it.

$$a \leftarrow not\ a, not\ b \qquad b \leftarrow not\ a, not\ b$$

We start the algorithm with all the possible negative hypotheses and no positive ones:

- $H_0^- = \{not\ a, not\ b\}, H_0^+ = \emptyset$.
- $P \cup H_0$ is not a consistent scenario.
- $H_0^+ = \emptyset$ is non-redundant.
- $H_1^+ = H_0^+$ is unavoidable.
- $P \cup H_1$ is not a consistent scenario.
- We non-deterministically choose one negative hypothesis $not\ L$ from $H_1^- = \{not\ a, not\ b\}$ such that $P \cup H_1 \vdash L$ and $P \cup H_1 \vdash not\ L$. In this case, both $not\ a$ and $not\ b$, so we can choose any one of them. Let us say we choose $not\ a$. Then $H_2^+ = H_1^+ \cup \{1\}$ and $H_2^- = H_1^- \setminus \{not\ a\}$. And we go on to step 3 again.
- H_2^+ is non-redundant.
- $H_3^+ = H_2^+$ is unavoidable.
- $P \cup H_3$ is a consistent scenario.
- $P \cup H_3$ is a 2-valued complete scenario, so $H_3 = H_3^+ \cup H_3^- = \{a\} \cup \{not\ b\} = \{a, not\ b\}$ is a Revision Complete Scenario and the algorithm terminates successfully.

If we were to choose $not\ b$ instead of $not\ a$ in step 12, the resulting Revision Complete Scenario would be $\{not\ a, b\}$. These Revision Complete Scenario coincide with those produced by the algorithm in definition 14.

Theorem 4. ***The sets*** $H = H^+ \cup H^-$ ***resulting from the execution of algorithm of definition 15 are the Revision Complete Scenarios***

Proof. Trivial, by construction of the algorithm. □

2.4 The Name of the Game

Why the name "Revision" Complete Scenarios? The "Revision" part of the name comes from the assumption revision we do when an assumption $not\ A \in H^-$ leads to a contradiction in P, i.e., $\big(P \cup H^- \vdash \{A, not\ A\}\big) \wedge \big(P \cup (H^- \setminus \{not\ A\}) \nvdash \{A, not\ A\}\big)$.

In such a case we accept to revise $not\ L$ to its positive counterpart L. This is the specific form of reasoning by *Reductio ad Absurdum* we take here: if adding $not\ A$ to P in the context of H^- leads to self inconsistency, then, by absurdity, we should assume A instead of $not\ A$. A becomes, thus, one of the positive hypotheses.

3 Syntactic Perspective of Revision Complete Scenarios over Normal Logic Programs

In [3] the authors proved that every Stable Model of a NLP corresponds to a 2-valued complete, consistent and admissible scenario. In [10] the author shows that when a NLP has no SMs it is because the Normal Logic Program has Odd Loops Over Negation (OLONs) and/or Infinite Chains Over Negation (ICONs), although the author does not employ these designations. These designations are taken from [14].

For the sake of readability and self-containment we briefly present some examples of OLONs and ICONs. Intuitively an OLON is a set of rules of a NLP which induce

a cycle over some literals in the dependency graph. The cycle of an OLON has the characteristic of having an Odd number of default Negated arcs around the cycle.

An example of an OLON is given in example 1. There we can see that the atom $intend\_we\_to\_invade$ is in a cycle across the dependency graph, and that along that cycle there is only 1 (an Odd number) default negation.

Another example of an OLON is present in example 2. There the atom $mountain$ is in a cycle with 3 default negations along the circular dependency graph. The same is true for $travel$ and $beach$.

The classical example of an ICON was first presented in [10]. It goes as follows:

$$p(X) \leftarrow p(s(X)) \qquad p(X) \leftarrow not\ p(s(X))$$

where X is a variable. The ground version of this program when there is only one constant 0 is the infinite program

$$\begin{array}{ll} p(0) \leftarrow p(s(0)) & p(0) \leftarrow not\ p(s(0)) \\ p(s(0)) \leftarrow p(s(s(0))) & p(s(0)) \leftarrow not\ p(s(s(0))) \\ p(s(s(0))) \leftarrow p(s(s(s(0)))) & p(s(s(0))) \leftarrow not\ p(s(s(s(0)))) \\ \vdots & \vdots \end{array}$$

This example in particular is the one to which every other possible variation of an ICON reduces to (proven in [10]). As it can be easily seen, there is an infinitely long chain of support for any $p(X)$ with an infinite number of default negations.

As we just said, in [10] the author proves that only OLONs and/or ICONs can prevent the existence of SMs in a NLP. Therefore, since our Revision Complete Scenario guarantee the Existence of a Model for any given NLP it follows that the Revision Complete Scenario deal with OLONs and ICONs in a way that the Stable Models semantics did not. This is achieved by means of the reasoning by *Reductio ad Absurdum* we explained in subsection 2.4.

4 Collaborative Argumentation

The classical perspective on Argumentation is typically of a competitive nature: there are arguments and counter-arguments, all of them attacking each other and struggling for admissibility. The ones which counter-attack all its attackers are admissible.

Typically, one takes one argument — a set of hypotheses H — and check if it is admissible, and if $P \cup H$ is a consistent scenario. If 2-valuedness is a requisite, then an extra test for 2-valued completeness is required.

We now generalize this approach in a constructive way, by building up a compromise Revision Complete Scenario starting from several conflicting 2-valued complete and consistent Models of P — each corresponding to an argument. This is what the algorithm below does.

First, we take all the conflicting models N_1, N_2, ..., N_n and calculate the set of all the possible positive hypotheses $M^+ = \bigcup_{i=1}^{n} N_i^+$; and the set of all the possible negative hypotheses $M^- = \bigcup_{i=1}^{n} N_i^-$. M^+ and M^- will now be used to guide the algorithm below in order to ensure consensus, i.e., the resulting Revision Complete

Scenario H will have no positive hypotheses outside M^+, nor will it have negative hypotheses outside M^-. The algorithm goes as follows:

Definition 16. ***Revision Complete Scenario*** H ***construction from conflicting models*** $N_1, N_2, \ldots N_n$

1. *Start with* $M = M^+ \cup M^-$. $M_0 = M$ *is inconsistent.*
2. $M_1^+ = M_0^+ \setminus \{L \in M_0^+ : not\ L \in M_0^-\}$, *and* $M_1^- = M_0^-$. M_1 *is now consistent.*
3. *If* M_i^- *is not weakly admissible then non-deterministically select one* L *such that* $not\ L \in M_i^-$, *there is an* E *such that* $E \rightsquigarrow L$, *and there is some* $not\ a \in E$ *such that* $P \cup M_i^- \cup E \nvdash a$. *Construct* $M_{i+1}^- = M_i^- \setminus \{not\ L\}$. *Repeat this step.*
4. *If* M_{i+1}^+ *is avoidable then* $M_{i+2}^+ = M_{i+1}^+ \setminus \{L\}$, *where* $P \cup (M_{i+1} \setminus \{L\}) \cup \{not\ L\}$ *is an inconsistent scenario.* $M_{i+2}^- = M_{i+1}^- \cup \{not\ L\}$ *only if* $L \in M^-$, *otherwise* $M_{i+2}^- = M_{i+1}^-$. *Go on to step 3 again.*
5. *If* $P \cup M_{i+2}$ *is not a consistent scenario then non-deterministically select one* L *such that* $P \cup M_{i+2} \vdash \{L, not\ L\}$, *and construct* $M_{i+3}^- = M_{i+2}^- \setminus \{not\ L\}$. *Go on to step 3 again.*
6. *If* $P \cup M_{i+2}$ *is not a 2-valued complete scenario then* $M_{i+3}^+ = M_{i+2}^+ \cup \{L\}$, *where* $P \cup M_{i+2} \nvdash L$ *and* $P \cup M_{i+2} \nvdash not\ L$ *and* $L \in M^+$, *and go on to step 4 again.*
7. $P \cup M_{i+2}$ *is a 2-valued complete and consistent scenario, where* M_{i+2}^+ *is non-redundant and unavoidable, and* M_{i+2}^- *is weakly admissible. By definition,* M_{i+2} *is a Revision Complete Scenario, therefore* $H = M_{i+2}$ *and the algorithm terminates successfully.*

□

In essence, this algorithm is a mixture of the Inconsistency Avoidance and Inconsistency Removal algorithms presented in subsection 2.3. We start with two sets M^+ and M^- containing, respectively, all the possible positive hypotheses that can be adopted in the final Revision Complete Scenario H, and all the possible negative hypotheses that can be adopted. Next, we remove from the set of positive hypotheses all those conflicting with the negative ones in order to ensure consistency. Now we need to ensure a weak admissibility of the current negative hypotheses M_i^-. For that we check if the M_i^- is weakly admissible, and if it is not, then we non-deterministically select and remove from M_i^- one of the negative hypotheses causing M_i^- failing to comply to this requirement. This step is repeated until weak admissibility is verified by M_i^-. Now we turn to the set of positive hypotheses M_i^+. If it is avoidable, then we non-deterministically select and remove from M_i^+ one positive hypothesis L which contributes to M_i^+ avoidability. We also add the correspondent default negation of that positive hypotheses $not\ L$ to M_i^-, but only if $not\ L$ was already in M^- — the initial set of all the adoptable negative hypotheses. This extra requirement ensures the final compromise Revision Complete Scenario H to be found is maximally compatible with all the initial models $N_1, N_2, \ldots, N_n$. When we add $not\ L$ to M_i^- we need to recheck its weak admissibility, so we go on to that step again. If M_i^+ was unavoidable, then we need to check it the whole $P \cup M_i$ is consistent. If this scenario fails consistency, then we remove from M_i^- one of the negative hypothesis whose positive counterpart was also being produced by $P \cup M_i$. Notice that when the resulting scenario is not consistent we

remove one inconsistency in favour of the positive hypotheses, since the presence of the correspondent negative produced the inconsistency. This is basically the mechanism of reasoning by *Reductio ad Absurdum* we use. Again we need to recheck the weak admissibility, so we go on to that step again. If the scenario $P \cup M_i$ was consistent, then we need to check if it is 2-valued complete. If it is not, then we non-deterministically select one adoptable positive hypothesis and add it to M_i^+. Now we need to recheck M_i^+'s unavoidability; so we go on to that step again. Finally, if $P \cup M_i$ was 2-valued complete then $H = M_i$ is a Revision Complete Scenario and the algorithm terminates successfully.

Example 11. **Example 2 revisited — A vacation problem** Recall the example 2 presented earlier. The program is:

$$travel \leftarrow not\ mountain \quad mountain \leftarrow not\ beach \quad beach \leftarrow not\ travel$$

Now assume that one of the friends going on vacation with the other two could not be present when they were getting together to decide their vacations' destinies. So, only John (the one who preferred going to the mountains, otherwise traveling it is), and Mary (she prefers going to the beach, otherwise going to the mountains is ok).

John's opinion is $J = \{mountain, not\ travel, not\ beach\}$, while Mary's choice is $Z = \{beach, not\ mountain, not\ travel\}$. We can already see that at least on one thing they agree: $not\ travel$. We now find the largest set of positive hypotheses we can consider $M^+ = J^+ \cup Z^+ = \{mountain, beach\}$ and the largest set of negative hypotheses we can consider $M^- = J^- \cup Z^- = \{not\ travel, not\ beach, not\ mountain\}$. And now the algorithm starts:

$M = M^+ \cup M^- = \{mountain, beach, not\ mountain, not\ beach, not\ travel\}$

Going through the steps of the algorithm we have:

- $M_0 = M$.
- $M_1^+ = M_0^+ \setminus \{mountain, beach\} = \emptyset$, $M_1^- = M_0^-$.
- M_1^- is not weakly admissible, so we non-deterministically select one L such that $not\ L \in M_1^-$ is one of the causes for M_1^- not complying to the weak admissibility condition: for example, $L = mountain$. $M_2^- = M_1^- \setminus \{not\ mountain\} = \{not\ beach, not\ travel\}$. We repeat this set and now we must remove $not\ beach$ from M_2^-. $M_3^- = M_2^- \setminus \{not\ beach\} = \{not\ travel\}$.
- $M_3^+ = M_2^+ = M_1^+ = \emptyset$ is unavoidable.
- $P \cup M_3$ is a consistent scenario.
- $P \cup M_3$ is not a 2-valued complete scenario. So $M_4^+ = M_3^+ \cup \{mountain\}$ because $mountain$ is the only literal which verifies $P \cup M_3 \nvdash mountain$ and $P \cup M_3 \nvdash not\ mountain$. Now we go on to step 4 of the algorithm again.
- M_4^+ is unavoidable.
- $P \cup M_4$ is consistent.
- $P \cup M_4$ is 2-valued complete, so $H = M_4^+ \cup M_4^- = \{mountain, not\ travel\}$ and the algorithm terminates successfully.

In the end, the resulting model is $least(P \cup H) = \{mountain, beach, not\ travel\}$. Notice that $beach$ is just a consequence of $not\ travel$ in P, it does not have to be a hypothesis. If other atoms were to be chosen at step 3 other alternative solutions would be found.

5 Integrity Constraints

Example 12. **Middle Region Politics Revisited** Recall the example 4 presented earlier. We are now going to add extra complexity to it.

We already know the two factions which are at odds and their thinking.

$$oppression \leftarrow not\ end\_of\_terrorism \quad end\_of\_terrorism \leftarrow oppression$$
$$terrorism \leftarrow not\ end\_of\_oppression \quad end\_of\_oppression \leftarrow terrorism$$

We now combine these two sets of rules with the two following Integrity Constraints (ICs) which guarantee that $oppression$ and $end\_of\_oppression$ are never simultaneously true; and the same happens with terror:

$$falsum \leftarrow oppression, end\_of\_oppression, not\ falsum$$
$$falsum \leftarrow terrorism, end\_of\_terrorism, not\ falsum$$

So far so good, there is still a single joint set of hypotheses resulting in a consistent scenario $\{end\_of\_oppression, end\_of\_terrorism\}$. Still, there is no SM for this program. But introducing either one or both of the next two rules, makes it impossible to satisfy the ICs:

$$oppression \leftarrow not\ terrorism \quad terrorism \leftarrow not\ oppression$$

In this case all the consistent and 2-valued complete scenarios contain the atom $falsum$. There are still no Stable Models for the resulting program. The semantics we propose allows two models for this program, which correspond to the 2-valued complete consistent scenarios, both containing $falsum$. We can discard them on this account or examine their failure to satisfy the ICs.

6 Conclusions and Future Work

We have managed to assign a complete 2-valued semantics to every Normal Logic Program, by employing an argumentation framework that readily extends the argumentation framework of Stable Models semantics. We also presented three algorithms for finding the Revision Complete Scenario of any Normal Logic Program. Every Stable Model of a Normal Logic Program corresponds to a Revision Complete Scenario and, in that sense, our algorithms allow for a different perspective on Stable Models semantics: any Stable Model can be seen as the result of an iterative process of Inconsistency Removal or Inconsistency Avoidance. In any case, Stable Models are the final result of such inconsistency removal/avoidance where any initial positive hypotheses remain in the end. In the process, we have extended argumentation with *Reductio ad Absurdum* reasoning for that purpose, and shown how Collaborative Argumentation can be defined in that context.

Future work concerns the extension to Generalized Logic Programs and Extended Logic Programs, and the seamless merging with more general belief revision in Logic Programs.

Some of the applications enabled by this improved semantics of Normal Logic Programs, concern the ability to guarantee that the meaning of diverse programs, e.g. arising from Semantic Web usage, always has a semantics. Similarly, we can also ensure this property whenever updating programs, including the case where an autonomous program evolves through self-updating [1]. Such applications will be enabled by the ongoing implementation.

Acknowledgments We deeply thank Robert A. Kowalski for his crucial help in clarifying our ideas and their presentation.

References

1. J. J. Alferes, A. Brogi, J. A. Leite, and L. M. Pereira. Evolving logic programs. In S. Flesca et al., editor, *JELIA*, volume 2424 of *LNCS*, pages 50–61. Springer, 2002.
2. J. J. Alferes and L. M. Pereira. An argumentation theoretic semantics based on non-refutable falsity. In J. Dix et al., editor, *NMELP*, pages 3–22. Springer, 1994.
3. A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.*, 93:63–101, 1997.
4. P. M. Dung. Negations as hypotheses: An abductive foundation for logic programming. In *ICLP*, pages 3–17. MIT Press, 1991.
5. P. M. Dung. An argumentation semantics for logic programming with explicit negation. In *ICLP*, pages 616–630. MIT Press, 1993.
6. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.
7. P. M. Dung, R. A. Kowalski, and F. Toni. Dialectic proof procedures for assumption-based, admissible argumentation. *Artif. Intell.*, 170(2):114–159, 2006.
8. P. M. Dung, P. Mancarella, and F. Toni. Argumentation-based proof procedures for credulous and sceptical non-monotonic reasoning. In *Computational Logic: Logic Programming and Beyond*, volume 2408 LNCS, pages 289–310. Springer, 2002.
9. P. M. Dung and T. C. Son. An argument-based approach to reasoning with specificity. *Artif. Intell.*, 133(1-2):35–85, 2001.
10. F. Fages. Consistency of Clark's completion and existence of stable models. *Methods of Logic in Computer Science*, 1:51–60, 1994.
11. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.
12. A. C. Kakas and P. Mancarella. Negation as stable hypotheses. In *LPNMR*, pages 275–288. MIT Press, 1991.
13. A. C. Kakas and F. Toni. Computing argumentation in logic programming. *J. Log. Comput.*, 9(4):515–562, 1999.
14. L. M. Pereira and A. M. Pinto. Revised stable models - a semantics for logic programs. In G. Dias et al., editor, *Progress in AI*, volume 3808 of *LNCS*, pages 29–42. Springer, 2005.
15. A. M. Pinto. Explorations in revised stable models — a new semantics for logic programs. Master's thesis, Universidade Nova de Lisboa, February 2005.
16. L. Soares. Revising undefinedness in the well-founded semantics of logic programs. Master's thesis, Universidade Nova de Lisboa, 2006.
17. F. Toni and R. A. Kowalski. An argumentation-theoretic approach to logic program transformation. In *LOPSTR*, volume 1048 of *LNCS*, pages 61–75. Springer, 1996.