Iterative Belief Revision in Extended Logic Programming[1]

Jia-Huai You
Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada, you@cs.ualberta.ca

Robert Cartwright
Department of Computer Science, Rice University
Houston, Texas, cork@cs.rice.edu

Ming Li
Department of Computer Science, University of Waterloo
Waterloo, Ontario, Canada, mli@math.uwaterloo.ca

Abstract Extended logic programming augments conventional logic programming with both default and explicit negation. Several semantics for extended logic programs have been proposed that extend the well-founded semantics for logic programs with default negation (called normal programs). We show that two of these extended semantics are intractable; both Dung's grounded argumentation semantics and the well-founded semantics of Alferes *et al.* are NP-hard. Nevertheless, we also show that these two semantics have a common core, a more restricted form of the grounded semantics, which is tractable and can be computed iteratively in quadratic time. Moreover, this semantics is a representative of a rich class of tractable semantics based on a notion of iterative belief revision.

1 Introduction

The semantics of logic programs with two kinds of negation, *default negation* and *explicit negation*, has been studied extensively in the recent literature (e.g. [1, 4, 9, 12, 16, 18, 19, 24, 25, 26, 27]). Logic programs with default and explicit negation are called *extended logic programs*. The added expressive power of extended logic programming has been exploited in a variety of domains, such as reasoning with actions [17, 27], diagnosis and program debugging [28], argumentation and political debate [9, 4], and reason maintenance [30].

[1] Some results have been reported in a preliminary version of this paper [32].

In a ground logic program with negation, a semantics maps a program to a set of assumptions of the form **not** ϕ where ϕ is a ground atom. There are two fundamentally different approaches to defining semantics of logic programs with negation: *skeptical* and *credulous*. A skeptical semantics defines a unique extension of the least model semantics for conventional logic programs, while a credulous semantics defines many possible maximal extensions. The skeptical approach captures the intuitive notion of cautious reasoning; the set of assumptions is usually constructed by an iterative, monotonic process [5].[2] This property, called "iterative constructibility" in the literature, is important because it means that programs can produce output incrementally; once an assumption is established, it is never retracted.

Some of the skeptical semantics proposed for extended programs are extensions of the well-founded semantics for normal programs [15] (see, for example, [1, 9, 24, 27, 30, 31]). The goal in general is to define a constructive, yet consistent semantics for extended programs. The semantics of normal programs is a well-understood subject. In particular, the well-founded semantics is contained in any reasonable semantics. Moreover, it can be tractably computed by an iterative, monotonic process. As a result, the well-founded semantics serves as a cautious, tractable core for other semantics of normal programs.

In contrast, the semantics of extended programs is still an active area of research. First, when explicit negation is involved, the concepts and techniques used in formulating semantics for normal programs become inadequate, since the resulting semantics may be inconsistent. An inconsistent semantics is hardly acceptable as a way to characterize the semantics of a program, since the notion of logical consequences in this case becomes trivial.[3] Secondly, logic programming with two kinds of negation is a nonmonotonic formalism whose application area largely lies in those problems where available information is often incomplete. In these problems, new information may cause contradiction. As a simple example, consider the fact that, normally, we expect lights become on when we turn on the switch. This can be expressed by the following clause:

$$lights\_on \leftarrow switch\_on, \mathbf{not}\ abnormal$$

But what if we turn on the switch and find there is no *light*? Clearly, the logic program that represents all this information should not be treated as inconsistent.

[2] The *ideal skeptical semantics* of Alferes et al. is a hybrid semantics that does not fit either of these models; it is unique but it is not constructed by a monotonic, iterative process. We discuss this semantics in detail in Section 5.

[3] But there are approaches to inconsistency without trivializing the notion of logic consequences; e.g. the *paraconsistent approach* [3].

The problem of resolving contradiction has been the main focus of the various semantics for extended programs. With regard to this goal, we believe one should be prepared to answer the following questions:

- Are the various semantics tractable and, if not, why?
- Do these semantics accommodate incremental computation of answers (iterative constructibility)?
- Is there a cautious core semantics for extended programs, analogous to the well-founded semantics for normal programs, that is tractable and accommodates incremental computation?

The tractability question is important because it determines whether there exists a feasible implementation for ground programs. Moreover, the tractability of a semantics for ground programs is a good indicator of the inherent complexity properties for non-ground programs.

Some of these questions have been investigated in [30, 31], where the authors show that the *contradiction removal semantics* [25] for extended logic programs is NP-hard [30, 31]. They also propose an alternative semantics based on *global* revision that is tractable. A global revision semantics consists of two distinct phases. First, it constructs the well-founded model for an extended program by treating explicitly negated atoms as distinct propositions. Second, it revises the well-founded model to eliminate contradictions. In a global revision semantics, no knowledge is available until a program is revised as a whole. Hence, it does not support incremental computation. Moreover, there is no clear relationship between this global revision semantics and other prominent semantics for extended logic programs, namely, Dung's grounded argumentation semantics and the well-founded semantics of Alferes *et al.*

In this paper we propose a new tractable semantics for extended logic programs that accommodates incremental computation and forms a common core for Dung's grounded argumentation semantics and Alferes et al.'s well-founded semantics. Our semantics is based on the concept of *iterative belief revision* where a reasoning agent incrementally establishes his beliefs by following an iterative process that builds on previous knowledge. The iterative process terminates when no new beliefs are generated. Once a belief is established, it can never be retracted. This type of belief revision has not been explicitly investigated in the literature. The only previous semantics that appears to be compatible with the notion of iterative belief revision is the grounded semantics proposed by Dung [9]. For this reason, our study of iterative belief revision begins with Dung's semantics.

Dung defines the meaning of a ground program as the least fixpoint of an acceptability operator $\mathcal{A}: \bigsqcup \mathcal{A}^i(\emptyset)$. To recast this semantics in the framework of iterative belief revision, we show that the acceptability operator $\mathcal{A}$ is the composition $\mathcal{R} \circ \mathcal{N}$ of two functions where $\mathcal{N}$ generates assumptions and $\mathcal{R}$ revises them to maintain consistency. The new assumptions generated by the acceptability operator $\mathcal{A} = \mathcal{R} \circ \mathcal{N}$ at each step of the iteration process used to compute $\bigsqcup \mathcal{A}^i(\emptyset)$ are guaranteed to appear in the least fixpoint and hence can be output immediately. The decomposition of $\mathcal{A}$ into $\mathcal{R} \circ \mathcal{N}$ is important because the two functions can be computed independently and results are combined.

Unfortunately, we show that it is NP-hard to compute Dung's grounded semantics. Similarly, we show that a different semantics, the well-founded semantics for extended programs proposed by Alferes et al. [1], is also NP-hard. The sources of the intractability in these two semantics are different: one is due to the restriction to *sound* arguments, and the other is due to retracting as few assumptions as possible, commonly known as the minimal change property. These insights lead us to discover a new grounded semantics, called *the common grounded semantics*, that is tractable and iteratively constructible. This semantics entails both Dung's and Alferes et al.'s in the sense that the assumption set constructed by it is a subset of those by the other two semantics. This property has two important implications. First, since there is no universally agreed semantics for logic programs with default as well as explicit negation, it is important to understand how some of the previously proposed semantics are related. The revealation of the relationship between Dung's semantics and the one by Alferes et al. is a major step towards this goal. This result is particularly interesting since it hinges upon the important property of tractability: Despite the fact that they are in general incompatible, they have an important, relatively substantial common subset. Second, the common grounded semantics provides an efficient mechanism to compute an upper-bound approximation to both of them.

We subsequently generalize the common grounded semantics to produce a framework for defining a rich class of grounded semantics based on iterative belief revision. Each of the semantics in this class uses a different choice for the functions $\mathcal{R}$ and $\mathcal{N}$ to generate assumptions and to perform belief revision. We give sufficient conditions for such a semantics to be "well-defined", tractable, and *sound* (in the sense that a non-contradictory program is always guaranteed a consistent semantics). We present two applications of this framework. First, we construct a more informative grounded semantics than the common grounded semantics by employing a less restrictive belief revision function. Second, we construct a tractable semantics for extended programs with *assumption-deriving clauses* (the clauses whose head is an assumption **not** ϕ). A clause of this form specifies that an assumption holds under the condition specified in the body of the clause. Since the so-called *coher-*

ence principle [23] can be specified by such clauses, the new semantics provides a tractable, coherent argumentation semantics for extended programs with integrity constraints.

The results presented here should be of potential interest to the nonmonotonic reasoning community as well, as Dung's argumentation framework is not restricted to logic programs only, but serves as a general formalism for nonmonotonic reasoning [11].

The next section presents the basic definitions for extended programs. Section 3 reformulates Dung's semantics in terms of belief revision and show that the grounded semantics is intractable. Section 4 introduces the common grounded semantics and shows it can be computed in quadratic time. In Section 5 we further show that the semantics proposed by Alferes et al. is also intractable, and the common grounded semantics is again an upper-bound approximation to it. Section 6 describes the general framework of iterative belief revision with applications. Additional related work is discussed in Section 7 with final remarks given in Section 8.

2 Background

We assume a language that contains a collection H_L of *individuals*, called *(ground) atoms*, ϕ, φ,..., $\phi^{\sim}$, $\varphi^{\sim}$,...; and a collection $\mathbf{not}\,(H_L)$ of *assumptions* of the form $\mathbf{not}\,\phi$, $\phi \in H_L$. Any subset of $\mathbf{not}\,(H_L)$ is called *an assumption set.* Atoms of the form $\phi^{\sim}$ are *explicitly negated atoms.* We use this notation to emphasize that they are *syntactic* objects to start with. In addition, we assume in the language a distinct, special atom $\perp \notin H_L$, which denotes *contradiction* and may be used to express constraints.

A clause is of the form $\phi \leftarrow \varphi_1, ..., \varphi_k, \mathbf{not}\,\varphi_{k+1}, ..., \mathbf{not}\,\varphi_n$, where $k, n \geq 0$, $\phi \in H_L \cup \{\perp\}$, and φ_i's are individuals from H_L. When $\phi \in H_L$ it is called a *program clause*, and when $\phi = \perp$ it is called an *(integrity) constraint.* We sometimes denote a clause by $c \leftarrow \alpha, \beta$ where α denotes the set of the atoms in the body, and β the set of the assumptions.

A *normal program* is a set of program clauses, wherein individuals such as ϕ and $\phi^{\sim}$ are distinct propositions. An *extended program* (or just *a program*) is a set of program clauses and constraints. In extended logic programming, whenever we derive ϕ and $\phi^{\sim}$ we have a *contradiction.* This can be expressed by adding the set $\Gamma_\perp = \{\perp \leftarrow \phi, \phi^{\sim} \mid \phi, \phi^{\sim} \in H_L\}$. From now on, when we refer to a program P, we assume the constraints in $\Gamma_\perp$ are always included. Other constraints are possible. For example, if we do not want complementary assumptions, say $\mathbf{not}\,\phi$ and $\mathbf{not}\,\phi^{\sim}$, to be assumed at the same time, we can write $\perp \leftarrow \mathbf{not}\,\phi, \mathbf{not}\,\phi^{\sim}$.

Let P be a set of clauses and S be an assumption set. We define a derivation relation $\vdash_d$ as follows: $P \cup S \vdash_d c$ iff there is a clause $c \leftarrow \in P$, or inductively, there is a clause

$c \leftarrow \alpha, \beta \in P$ such that for each $\rho \in \alpha \cup \beta$, either $P \cup S \vdash_d \rho$ or ρ is an assumption in S.

A program P is said to be *non-contradictory* iff $P \not\vdash_d \bot$.

In the sequel, if not specified otherwise, P denotes a fixed but arbitrary extended program. Dung's notion of *argumentation framework* consists of a program P, a set $AS \subseteq \mathbf{not}\,(H_L)$ of assumptions of interest, and two *attack* relations. We will mention P only and specify attack relations separately. The default AS is $\mathbf{not}\,(H_L)$, but very often we are interested only in the set of the assumptions that appear in the bodies of the clauses in P, denoted $\mathbf{not}\,(P)$, since these are the assumptions that may be used to carry out additional derivations. In particular, this is the convention we adopt when we use examples for the illustration purposes.

3 The Grounded Argumentation Semantics

An *argument* is a set of assumptions. The argumentation semantics of Dung [9, 10] is based on the intuitive notions of *attacks* on arguments, and *acceptability* of arguments.

Definition 3.1 An argument A is said to be *sound*, w.r.t. to a program P, iff there exists no non-contradictory subset P' of P such that $P' \cup A \vdash_d \bot$. $\square$

Note that an argument being sound is independent of whether P is contradictory or not. This treatment is more general than that of Dung's, since the notion of soundness will not become trivialized in case of contradictory programs. Consider, for example,

$$P = \{a \leftarrow;\ a^{\sim} \leftarrow;\ b \leftarrow \mathbf{not}\, c;\ b^{\sim} \leftarrow \mathbf{not}\, d\}.$$

The arguments $\{\mathbf{not}\, c\}$ and $\{\mathbf{not}\, d\}$ are both sound, though the argument $\{\mathbf{not}\, c,\ \mathbf{not}\, d\}$ is unsound.

The following proposition immediately follows from this definition of soundness.

Proposition 3.1 Given a program P, (i) the empty argument is always sound; and (ii) if an argument A is sound, then any $B \subseteq A$ is also sound. $\square$

The central question is when an argument is considered *acceptable*. This is based on two kinds of attack relations between arguments.

Definition 3.2 Let P be a program and A and B be sound arguments.

A is said to be *RAA-attacked* (*reductio ad absurdum*) by B if $P \cup A \cup B \vdash_d \bot$.

A is said to be *g-attacked* (*groundly attacked*) by B if $P \cup B \vdash_d \xi$, for some $\mathbf{not}\, \xi \in A$.

A is said to be *attacked* by B if it is either RAA-attacked by B or g-attacked by B. $\square$

Definition 3.3 A sound argument A is *acceptable* w.r.t. an assumption set S iff for any sound argument B that attacks it, B is g-attacked by a sound subset of S. $\Box$

The notion of acceptability can be used to define a function, Dung's acceptability function $\mathcal{D}_P$, over sets of assumptions as:

$$\mathcal{D}_P(S) = \{\mathbf{not}\,\phi \mid \mathbf{not}\,\phi \in A,\ A \text{ is acceptable w.r.t. } S\}.$$

It is easy to verify that the function $\mathcal{D}_P$ is *monotonic*, i.e., from $S_1 \subseteq S_2$ we have $\mathcal{D}_P(S_1) \subseteq \mathcal{D}_P(S_2)$. Hence according to the fixpoint theory, commonly referred to as the Tarski-Knaster fixpoint theorem [29], we know $\mathcal{D}_P$ has a *least fixpoint* over the domain consisting of all the subsets of $\mathbf{not}\,(H_L)$. This least fixpoint of $\mathcal{D}_P$ is defined as the *grounded (skeptical) semantics* of P.

Note that the fixpoint theory is applicable to non-ground programs (i.e., programs with variables). This means that the semantics can be defined easily for these programs without the need of instantiating them. However, we also know from the fixpoint theory that only if the function $\mathcal{D}_P$ is *continuous*, the least fixpoint can be computed iteratively from the smallest element in the domain. It is known that this is the case for finite propositional programs, or when the assumption set AS of interest is finite. In the sequel, if not said otherwise we assume AS is finite.

3.1 Grounded semantics as iterative revision

In this section we provide a reformulation of Dung's grounded semantics in terms of iterative belief revision.

First, considering the acceptability of arguments instead of individual assumptions is disastrous computationally, since the number of arguments is exponential to the number of assumptions. Here we show that the acceptability of an argument coincides with the acceptability of all the individual assumptions in it. This result is essential in relating argumentation-based semantics with non-argumentation-based semantics.

Theorem 3.2 *Given a program P, a sound argument A is acceptable w.r.t. an assumption set S iff for each $\mathbf{not}\,\phi \in A$, $\{\mathbf{not}\,\phi\}$ is acceptable w.r.t. S.*

Proof: ($\Leftarrow$) Assume for any $\mathbf{not}\,\phi \in A$, $\{\mathbf{not}\,\phi\}$ is acceptable w.r.t. S and show A is acceptable w.r.t. S.

Let $\mathbf{not}\,\phi \in A$. First consider the case that $\{\mathbf{not}\,\phi\}$ is acceptable because for any sound argument B, $P \cup B \vdash_d \phi$ implies $P \cup S' \vdash_d \xi$, for some $\mathbf{not}\,\xi \in B$, where $S' \subseteq S$ is sound.

Clearly, this trivially holds for A, since $\mathbf{not}\,\phi \in A$. Thus we only need to show (a) A is sound, and (b) for any sound argument B that RAA-attacks A, a sound subset $S' \subseteq S$ g-attacks B.

To prove (a), for the sake of contradiction, let us assume that A is not sound, i.e., there exists a non-contradictory subprogram P' such that $P' \cup A \vdash_d \bot$. Since $\{\mathbf{not}\,\phi\}$ is sound, there exists a minimal $A' \subseteq A$ such that $P' \cup A' \not\vdash_d \bot$ but $P' \cup A' \cup \{\mathbf{not}\,\phi\} \vdash_d \bot$. Since $\{\mathbf{not}\,\phi\}$ is acceptable, there exists $\mathbf{not}\,\xi \in A'$, and hence $\mathbf{not}\,\xi \in A$, such that there is a sound $S' \subseteq S$ such that $P \cup S' \vdash_d \xi$. This implies that $\{\mathbf{not}\,\xi\}$ is not acceptable, contradicting the hypothesis that every assumption in A is acceptable.

To prove (b) we consider an arbitrary sound argument B such that $P \cup A \cup B \vdash_d \bot$. Let $A' \subseteq A$ and $B' \subseteq B$ be such that $C = B' \cup (A' - \{\mathbf{not}\,\phi\})$ is sound but $C \cup \{\mathbf{not}\,\phi\}$ is not. By the assumption that $\mathbf{not}\,\phi$ is acceptable, there exists $\mathbf{not}\,\xi \in C$ such that $P \cup S \vdash_d \xi$. Since every assumption is acceptable w.r.t. S individually, $\mathbf{not}\,\xi \notin A$. It follows that $\mathbf{not}\,\xi \in B$. This shows A is acceptable w.r.t. S.

($\Rightarrow$) Similar but simpler. It is thus omitted. $\square$

Due to this theorem we are able to reformulate the notion of acceptability.

Definition 3.4 An assumption $\mathbf{not}\,\phi$ is *acceptable* w.r.t. an assumption set S iff for any sound argument B that attacks $\{\mathbf{not}\,\phi\}$, B is g-attacked by a sound subset of S. $\square$

Note that if $\{\mathbf{not}\,\phi\}$ is acceptable, then $\{\mathbf{not}\,\phi\}$ must be sound, and hence the condition of $\{\mathbf{not}\,\phi\}$ being sound is dropped.

Due to Theorem 3.2, the function $\mathcal{D}_P$ can be redefined as:

$$\mathcal{D}_P(S) = \{\mathbf{not}\,\phi \mid \mathbf{not}\,\phi \text{ is acceptable w.r.t. } S\}.$$

Further, as we will see, there is a special interest in splitting this function into two parts.

Definition 3.5 Define the following functions over sets S of assumptions (where B is a sound argument and S' is a sound subset of S):

$$\mathcal{DN}_P(S) = \{\mathbf{not}\,\phi \mid \forall B, B \text{ g-attacks } \{\mathbf{not}\,\phi\} \Rightarrow \exists S', S' \text{ g-attacks } B\}$$
$$\mathcal{DR}_P(S) = \{\mathbf{not}\,\phi \mid \forall B, B \ \mathit{RAA}\text{-attacks } \{\mathbf{not}\,\phi\} \Rightarrow \exists S', S' \text{ g-attacks } B\}$$

$\square$

The following fact is easy to verify.

Lemma 3.1 $\mathcal{D}_P(S) = \mathcal{DR}_P(S) \cap \mathcal{DN}_P(S)$. $\square$

Hence to compute $\mathcal{D}_P(S)$ one can compute $\mathcal{DN}_P(S)$ and $\mathcal{DR}_P(S)$ independently and then take the intersection. The function $\mathcal{D}_P$ can be seen as a composite function in the following way. Consider the definition below:

$$\mathcal{C}_P(N, S) = \{\mathbf{not}\ \phi \mid \mathbf{not}\ \phi \in N \text{ and } \mathbf{not}\ \phi \in \mathcal{DR}_P(S)\}.$$

With this definition, it is easy to check that the following equation holds:

$$\mathcal{D}_P(S) = \mathcal{C}_P(\mathcal{DN}_P(S), S).$$

The function $\mathcal{C}_P$ can be defined equivalently by a unary function $\mathcal{DR}_{P,S}$ (called a *Curried function*). We then can define $\mathcal{D}_P$ as:

$$\mathcal{D}_P = \mathcal{DR}_{P,S} \circ \mathcal{DN}_P$$

Based on this equation, we can interpret the notion of acceptability as belief revision: $\mathcal{DN}_P$ generates assumptions which are then *revised* by the operator $\mathcal{DR}_{P,S}$. This differs from almost all the previous approaches to belief revision, either in the context of propositional theories (e.g. [13]), or in the context of extended logic programming and RMS (e.g. [7, 25, 30]), where a revision operation is performed after the entire theory is determined contradictory. For a knowledge system with a large collection of data, possibly with frequent update requests, this is a formidable task. In contrast, revision in terms of the operator $\mathcal{DR}_{P,S} \circ \mathcal{DN}_P$ is performed *on the fly*. We will have more to say about this later in the paper.

Example 3.1 Let $P = \{a^{\sim} \leftarrow \mathbf{not}\ b;\ b \leftarrow \mathbf{not}\ d;\ e \leftarrow \mathbf{not}\ a;\ a \leftarrow \mathbf{not}\ c\}$. The grounded semantics (w.r.t. to $\mathbf{not}\ (P)$) can be calculated by:

$$\begin{aligned}
&S_0 = \emptyset\\
&S_1 = \mathcal{DR}_P(S_0) \cap \mathcal{DN}_P(S_0) = \{\mathbf{not}\ d, \mathbf{not}\ a\} \cap \{\mathbf{not}\ d, \mathbf{not}\ c\} = \{\mathbf{not}\ d\}\\
&S_2 = \mathcal{DR}_P(S_1) \cap \mathcal{DN}_P(S_1) = \{\mathbf{not}\ d, \mathbf{not}\ a, \mathbf{not}\ c\} \cap \{\mathbf{not}\ d, \mathbf{not}\ c\}\\
&\quad = \{\mathbf{not}\ d, \mathbf{not}\ c\}\\
&S_3 = \mathcal{DR}_P(S_2) \cap \mathcal{DN}_P(S_2) = S_2
\end{aligned}$$

Or, equivalently by

$$\begin{aligned}
&S_0 = \emptyset\\
&S_1 = \mathcal{DR}_{P,S_0} \circ \mathcal{DN}_P(S_0) = \mathcal{DR}_{P,S_0}(\{\mathbf{not}\ d, \mathbf{not}\ c\}) = \{\mathbf{not}\ d\}\\
&S_2 = \mathcal{DR}_{P,S_1} \circ \mathcal{DN}_P(S_1) = \mathcal{DR}_{P,S_1}(\{\mathbf{not}\ d, \mathbf{not}\ c\}) = \{\mathbf{not}\ d, \mathbf{not}\ c\}\\
&S_3 = \mathcal{DR}_{P,S_2} \circ \mathcal{DN}_P(S_2) = S_2
\end{aligned}$$

$\square$

3.2 The grounded semantics is intractable

Unfortunately, computing the grounded semantics is NP-hard, since it is NP-complete to determine whether an atom is derivable from a sound argument.

Theorem 3.3 *Give a program P and an atom ϕ, it is NP-complete to determine if there is a sound argument B such that $P \cup B \vdash_d \phi$.*

Proof: First, it is straightforward to check that the problem is in NP. A nondeterministic Turing machine can simply guess an argument B, and then deterministically verify in polynomial time whether ϕ can be derived from $P \cup B$.

Next, we show the problem is NP-hard by a polynomial-time Turing-reduction from the well-known 3-SAT problem to it. Let (U, C) be an instance of 3-SAT where U is the set of variables (propositional symbols) and C a set of clauses. Assume there are n clauses in (U, C), where $n \geq 1$. The goal is to find an assignment of variables in U such that all the clauses in C are satisfied. For this purpose, we construct a program P from (U, C) as follows (an example is provided in Appendix): For each clause $x_i \vee y_i \vee z_i \in C$, put three program clauses into P:

$$c_i \leftarrow \textbf{not}\, x_i'$$
$$c_i \leftarrow \textbf{not}\, y_i'$$
$$c_i \leftarrow \textbf{not}\, z_i'$$

where c_i is a new symbol, and $x_i' = x_i^{\sim}$ if x_i is an atom and x_i' is the atomic part of x_i if x_i is a negated atom (i.e. if x_i is $\neg a$ then x_i' is a); similarly for y_i' and z_i'.

In addition, P has the program clause

$$\phi \leftarrow c_1, ..., c_n$$

and for each $x \in U$, P has a constraint

$$\perp \leftarrow \textbf{not}\, x, \textbf{not}\, x^{\sim}$$

Assume B is a sound argument such that $P \cup B \vdash_d \phi$.

Since the only way to derive $\perp$ is by using a constraint clause $\perp \leftarrow \textbf{not}\, x, \textbf{not}\, x^{\sim}$, B being sound implies there is no variable x for which both $\textbf{not}\, x$ and $\textbf{not}\, x^{\sim}$ are in B. Thus, B induces a partial, consistent interpretation I_B such that $I_B(x) = \mathit{true}$ if and only if $\textbf{not}\, x^{\sim} \in B$. Note that since $P \cup B \vdash_d \phi$, every clause in U is satisfied already by I_B. Since the remaining variables can be assigned arbitrarily, a model exists, i.e., (U, C) is satisfiable.

On the converse, if (U, C) is satisfiable, then there is a model, say M. Let S_M be the set of assumptions corresponding to M. That is, if $M(x) = \mathit{true}$ then $\textbf{not}\, x^{\sim} \in S_M$ and if $M(x) = \mathit{false}$ then $\mathbf{not}\, x \in S_M$. The fact that every clause is satisfied by M implies $P \cup S_M \vdash_d \phi$. Since M is a model and thus no variable x is both true and false in M. It follows that S_M does not contain both $\mathbf{not}\, x$ and $\mathbf{not}\, x^{\sim}$. Thus, $P \cup S_M \not\vdash_d \bot$, i.e. S_M is a sound argument.

Therefore, there exists a sound argument B such that $P \cup B \vdash_d \phi$ iff (U, C) is satisfiable.
□

4 A Tractable Grounded Semantics

We have shown that requiring an argument to be sound in attacks and counterattacks renders the semantics intractable. Then, a natural question is whether there exists a restricted form of grounded semantics, which is efficient to compute, and can be considered as an upper-bound approximation to the grounded semantics.

In this section, we show that such a semantics can be obtained by dropping the soundness requirement on arguments. We revise the notions of attack and acceptability and define a variant of the grounded semantics, called the *common grounded semantics (CGS)*. More important, we show that the common grounded semantics is computationally attractive since it can be computed in quadratic time.

4.1 The common grounded semantics

We first redefine the *RAA*-attack relation so that attacks can be captured locally. At the same time, the domain of attacks is expanded to include unsound arguments as well. In the following definition, we focus on attacks on individual assumptions.

Definition 4.1 Let P be a program and B an argument. An argument $\{\mathbf{not}\, \phi\}$ is said to be *RAA-attacked* by B iff there exist subsets $P' \subseteq P$ and $B' \subseteq B$, such that $P' \cup \{\mathbf{not}\, \phi\} \cup B'$ is a *minimal* set satisfying $P' \cup \{\mathbf{not}\, \phi\} \cup B' \vdash_d \bot$. □

The minimality requirement ensures that each assumption in $\{\mathbf{not}\, \phi\} \cup B'$ and each clause in P' is necessarily involved in a derivation of $\bot$. For example, with

$$P = \{a \leftarrow \mathbf{not}\, p;\ b \leftarrow \mathbf{not}\, c;\ b^{\sim} \leftarrow \mathbf{not}\, d\},$$

$\{\mathbf{not}\, p\}$ is not *RAA*-attacked by $B = \{\mathbf{not}\, c, \mathbf{not}\, d\}$, since for no $P' \subseteq P$, $P' \cup \{\mathbf{not}\, p\} \cup B$ is a minimal set satisfying $P' \cup \{\mathbf{not}\, p\} \cup B \vdash_d \bot$. In fact, the only such minimal set is $P'' \cup B$ where $P'' = \{b \leftarrow \mathbf{not}\, c;\ b^{\sim} \leftarrow \mathbf{not}\, d\}$.

The definition of *g-attack* is the same as before except that when we say A g-attacks B, we no longer insist that A and B be sound.

Definition 4.2 Let P be a program and S an assumption set. Define $\mathcal{A}_P(S) = \mathcal{N}_P(S) \cap \mathcal{R}_P(S)$ where (with B being an arbitrary argument)

$$\mathcal{N}_P(S) = \{\mathbf{not}\,\phi \mid \text{ for any } B, B \text{ g-attacks } \{\mathbf{not}\,\phi\} \Rightarrow S \text{ g-attacks } B\}$$
$$\mathcal{R}_P(S) = \{\mathbf{not}\,\phi \mid \text{ for any } B, B\ RAA\text{-attacks } \{\mathbf{not}\,\phi\} \Rightarrow S \text{ g-attacks } B\}$$

□

The following claim is easy to verify.

Lemma 4.1 The operator $\mathcal{A}_P$ is monotonic. That is, for any assumption sets S_1 and S_2, if $S_1 \subseteq S_2$ then $\mathcal{A}_P(S_1) \subseteq \mathcal{A}_P(S_2)$. □

Definition 4.3 The common grounded semantics (CGS) is defined by the least fixpoint of the acceptability operator $\mathcal{A}_P$. □

As shown in the following proposition, the assumption set constructed by CGS is a subset of the one by Dung's grounded semantics.

Proposition 4.1 *Let W be the least fixpoint of the operator $\mathcal{A}_P$ and W' be the least fixpoint of the operator $\mathcal{D}_P$. Then $W \subseteq W'$.*

Proof: By a straightforward induction on the fixpoint iterations that construct W and W'. □

By allowing unsound arguments to attack, the semantics changes in a subtle way. For any unsound argument B that attacks A, if there is a sound subset of B that also attacks A, then nothing changes. Thus the only situation where CGS may deviate from Dung's original grounded semantics is when there is an argument B which is *necessarily* involved in a derivation of an atom ϕ as well as in a derivation of $\perp$.

There are two situations in which the use of an unsound argument B makes difference: (i) the derivations of a pair of complementary literals ϕ and $\phi^{\sim}$ depend on B, and (ii) the derivation of one of them depends on B and the other is derivable without resorting to any assumptions. These two situations can best be explained using the following two examples.

Example 4.1 Consider

$$P = \{a \leftarrow \mathbf{not}\,b;\ d \leftarrow \mathbf{not}\,b^{\sim};\ b \leftarrow \mathbf{not}\,c;\ b^{\sim} \leftarrow \mathbf{not}\,c\}.$$

Since the only way to derive b, or $b^{\sim}$, is by using the unsound argument $\{\mathbf{not}\,c\}$, both assumptions $\mathbf{not}\,b$ and $\mathbf{not}\,b^{\sim}$ are acceptable in Dung's semantics but neither is in CGS. □

In this example, one may argue that, since Dung's attack relation is defined over sound arguments, it does not distinguish unsound arguments from "unsound" program clauses. When in doubt on whether there is something wrong with the argument or with the program fragment, the reasoner specified by CGS becomes conservative by allowing more attacking arguments and hence accepts less assumptions.

Example 4.2 Consider

$$P = \{a \leftarrow;\ a^\sim \leftarrow \mathbf{not}\, b;\ d \leftarrow \mathbf{not}\, a^\sim\}.$$

The grounded semantics is $\{\mathbf{not}\, a^\sim\}$ and CGS is $\emptyset$. $\Box$

This example illustrates that CGS may miss reasonable assumptions since it does not single out those unsound arguments that can help derive an atom, which is contradictory to what can be derived without using any assumptions. Indeed, the reason that the assumption $\mathbf{not}\, a^\sim$ above is reasonable (and in fact desirable) is because a is derivable using clauses involving no assumptions. This is in fact a special case of *coherence* [23], which can be enforced by clauses such as $\mathbf{not}\, \phi^\sim \leftarrow \phi$. This will be treated in Subsection 6.2 where we will show that these assumptions can be preserved without causing the semantics to be intractable.

The main interest in CGS lies in the fact that when unsound arguments are allowed to attack, the computation becomes regular and hence much simpler. This is to be shown in the rest of this section.

4.2 Computing the common grounded semantics

4.2.1 Computing $\mathcal{N}_P$

We define a function $\mathcal{F}_P$ over sets S of assumptions as:

$$\mathcal{F}_P(S) = \{\mathbf{not}\, \phi \mid P \cup S \not\vdash_d \phi\}.$$

Denote the function composition $\mathcal{F}_P \circ \mathcal{F}_P$ by $\mathcal{F}_P^2$.[4]

Theorem 4.2 $\mathcal{F}_P^2(S) = \mathcal{N}_P(S)$.

Proof: (i) $\mathcal{F}_P^2(S) \subseteq \mathcal{N}_P(S)$.

Assume $\mathbf{not}\, \phi \in \mathcal{F}_P^2(S)$. This means $P \cup \mathcal{F}_P(S) \not\vdash_d \phi$. Let B be any argument such that $P \cup B \vdash_d \phi$. Clearly, $B \neq \emptyset$. Then, by the definition of $\mathcal{F}_P$, we know $B - \mathcal{F}_P(S) \subseteq$

[4]This operator has been studied previously by Van Gelder [14] and Baral and Subrahmanian [2].

$\{\mathbf{not}\,\xi \mid P \cup S \vdash_d \xi\}$. We claim $B - \mathcal{F}_P(S) \neq \emptyset$. As otherwise $B \subseteq \mathcal{F}_P(S)$, and from $P \cup B \vdash_d \phi$ we get $P \cup \mathcal{F}_P(S) \vdash_d \phi$, resulting in a contradiction. Therefore, there exists at least one $\mathbf{not}\,\xi \in B$, $P \cup S \vdash_d \xi$. By the definition of $\mathcal{N}_P$, we know $\mathbf{not}\,\phi \in \mathcal{N}_P(S)$.

(ii) $\mathcal{N}_P(S) \subseteq \mathcal{F}_P^2(S)$.

Assume $\mathbf{not}\,\phi \in \mathcal{N}_P(S)$. This means, if $P \cup B \vdash_d \phi$ for any argument B, we have $P \cup S \vdash_d \xi$, for some $\mathbf{not}\,\xi \in B$. If for any argument B, $P \cup B \not\vdash_d \phi$, then clearly $P \cup \mathcal{F}_P(S) \not\vdash_d \phi$, and thus $\mathbf{not}\,\phi \in \mathcal{F}_P^2(S)$. Now let B be any argument such that $P \cup B \vdash_d \phi$ and consider each $B' \subseteq B$ such that B' is a minimal set satisfying $P \cup B' \vdash_d \phi$. Since $\mathbf{not}\,\phi \in \mathcal{N}_P(S)$, we have $P \cup S \vdash_d \xi$, for some $\mathbf{not}\,\xi \in B'$. Hence $\mathbf{not}\,\xi \notin \mathcal{F}_P(S)$. It follows $P \cup \mathcal{F}_P(S) \not\vdash_d \phi$. Hence, by definition, $\mathbf{not}\,\phi \in \mathcal{F}_P^2(S)$. $\Box$

Let n_1 be the number of atom occurrences in P and n_2 be the size of $AS \subseteq \mathbf{not}\,(H_L)$. Let $n = n_1 + n_2$ denote the input size. It is easy to see that computing $\mathcal{F}_P(S)$ is bounded by $O(n)$, as each clause needs to be used at most once and computing set complement is bounded by $O(n)$. Hence computing $\mathcal{F}_P^2(S)$ is still bounded by $O(n)$. It follows from Theorem 4.2 that

Lemma 4.2 Computing $\mathcal{N}_P(S)$ is bounded by $O(n)$. $\Box$

4.2.2 Computing $\mathcal{R}_P$

In the definition of $\mathcal{R}_P(S)$, an assumption $\mathbf{not}\,\phi$ is acceptable if every argument B *RAA*-attacking it is g-attacked by S. Computing acceptable assumptions by enumerating all the arguments obviously takes exponential time. Here we provide an alternative definition of $\mathcal{R}_P(S)$ and show its computation takes linear time.

The idea is to compute the set of the assumptions that *cannot* be accepted. Intuitively, each *RAA*-attack is the result of some derivation that leads to $\bot$, using some clauses and the set D of assumptions in them. That is, $P' \cup D$ is minimal, where $P' \subseteq P$, satisfying $P' \cup D \vdash_d \bot$. When we consider the acceptability of an assumption $\mathbf{not}\,\xi$ w.r.t. S, if $\mathbf{not}\,\xi$ is in such a D, then $\mathbf{not}\,\xi$ is *RAA*-attacked by $B = D - \{\mathbf{not}\,\xi\}$ (and any superset of it). If each such D is g-attacked by S, then $\mathbf{not}\,\xi$ is acceptable. Hence $\mathbf{not}\,\xi$ is not acceptable if one of those D's is not g-attacked by S. This may be phrased more precisely as: *any assumption* $\mathbf{not}\,\phi$ *that cannot be accepted due to an RAA-attack by argument B must be involved in a derivation of* $\bot$ *that uses no assumptions* $\mathbf{not}\,\xi \in B$ *such that* $P \cup S \vdash_d \xi$. If we can identify this set, its complement is then the result of $\mathcal{R}_P(S)$.

We first describe a program reduction that narrows the search for unacceptable assumptions to a subset of program P.

Definition 4.4 The *reduced program* P_S, generated from program P by an assumption set S, is defined as:

$$P_S = \{\phi \leftarrow \alpha, \beta \in P \mid P \cup S \not\vdash_d \xi, \text{ for each } (\mathbf{not}\,\xi) \in \beta\}. \quad \Box$$

Next we identify the set of the assumptions that are not acceptable.

Definition 4.5 Let P_S be a program obtained from program P by S. The *support* of ϕ w.r.t. P_S, denoted $\mathcal{S}_{P_S}(\phi)$, is defined inductively as: $\mathcal{S}_{P_S}(\phi)$ is the smallest set satisfying, for each clause $\phi \leftarrow \alpha, \beta \in P_S$, if $P_S \cup \mathbf{not}\,(P_S) \vdash_d \xi$, for all $\xi \in \alpha$, then $\beta \subseteq \mathcal{S}_{P_S}(\phi)$, and inductively, $\mathcal{S}_{P_S}(\xi) \subseteq \mathcal{S}_{P_S}(\phi)$, for each $\xi \in \alpha$. $\Box$

The following fact is easy to verify.

Proposition 4.3 If $\mathcal{S}_{P_S}(\bot) \neq \emptyset$, then $P_S \cup \mathcal{S}_{P_S}(\bot) \vdash_d \bot$. $\Box$

We now show that, under RAA-attack, the set of the assumptions that are acceptable is exactly the complement of $\mathcal{S}_{P_S}(\bot)$. Denote this complement by $\overline{\mathcal{S}_{P_S}(\bot)}$.

Theorem 4.4 $\mathcal{R}_P(S) = \overline{\mathcal{S}_{P_S}(\bot)}$.
Proof: (i) We show $\mathcal{R}_P(S) \subseteq \overline{\mathcal{S}_{P_S}(\bot)}$. Assume $\mathbf{not}\,\phi \in \mathcal{R}_P(S)$. From this assumption we know that for any argument B, if $P' \cup \{\mathbf{not}\,\phi\} \cup B'$, where $P' \subseteq P$ and $B' \subseteq B$, is minimal satisfying $P' \cup \{\mathbf{not}\,\phi\} \cup B' \vdash_d \bot$, then $P \cup S \vdash_d \xi$, for some $\mathbf{not}\,\xi \in B$. Since B' is also such an argument RAA-attacking $\{\mathbf{not}\,\phi\}$, we have $\mathbf{not}\,\xi \in B'$. Recall that P_S is obtained from P by removing clauses whose body contains any $\mathbf{not}\,\xi$ such that $P \cup S \vdash_d \xi$. This implies that each derivation of $\bot$ through $\mathbf{not}\,\phi$ in P has been made impossible in P_S. As a result, $\mathbf{not}\,\phi$ cannot participate in a derivation of $\bot$ in P_S. That is, $\mathbf{not}\,\phi \notin \mathcal{S}_{P_S}(\bot)$.

(ii) Show $\overline{\mathcal{S}_{P_S}(\bot)} \subseteq \mathcal{R}_P(S)$. For this purpose, we assume $\mathbf{not}\,\phi \notin \mathcal{R}_P(S)$ and show $\mathbf{not}\,\phi \in \mathcal{S}_{P_S}(\bot)$. By this assumption, there exists an argument B such that $P' \cup \{\mathbf{not}\,\phi\} \cup B$, where $P' \subseteq P$, is a minimal set satisfying $P' \cup \{\mathbf{not}\,\phi\} \cup B \vdash_d \bot$, and for any $\mathbf{not}\,\xi \in B$, $P \cup S \not\vdash_d \xi$. This implies that, for each clause $\varphi \leftarrow \alpha, \beta \in P'$, $\beta \subseteq B$, and consequently for any $\mathbf{not}\,\xi \in \beta$, $P \cup S \not\vdash_d \xi$. Then by definition, we have $P'_S = P'$, and hence $P'_S \cup \{\mathbf{not}\,\phi\} \cup B$ is still minimal satisfying $P'_S \cup \{\mathbf{not}\,\phi\} \cup B \vdash_d \bot$. This leads to $\{\mathbf{not}\,\phi\} \cup B \subseteq \mathcal{S}_{P_S}(\bot)$ which implies $\mathbf{not}\,\phi \in \mathcal{S}_{P_S}(\bot)$. $\Box$

Lemma 4.3 Computing $\mathcal{R}_P(S)$ is bounded by $O(n)$.
Proof: Due to Theorem 4.4, we only need to consider the problem of computing $\overline{\mathcal{S}_{P_S}(\bot)}$. First, reducing P to P_S takes linear time since each clause in P needs to be used at most once. For exactly the same reason, computing $\mathcal{S}_{P_S}(\bot)$ also takes linear time. Finally, computing the complement $\overline{\mathcal{S}_{P_S}(\bot)}$ is linear. Thus, computing $\mathcal{R}_P(S)$ is bounded by $O(n)$. $\Box$

Theorem 4.5 *Computing the common grounded semantics is bounded by $O(n^2)$.*
Proof: The least fixpoint of the operator $\mathcal{A}_P$ can be computed in at most $O(n)$ steps. It follows from Lemmas 4.2 and 4.3, and the fact that computing the intersection is linear, CGS can be computed in $O(n^2)$ time. $\Box$

5 CGS as a Tractable Approximation to WFS0

An interesting aspect of CGS is that it is also a tractable approximation to other semantics. In this section, we show this for an important semantics recently introduced by Alferes et al. [1].

Alferes et al. first introduced the *ideal skeptical semantics* (*ISS*) based on the elegant idea that the best *scenario* is the maximal one that admits (in the sense of set inclusion) any other admissible scenario. Then, the grounded ISS, called WFS0, is defined by an iteratively constructed subset of ISS. These semantics are enforced by the coherence principle, which says whenever we can derive ϕ, we should assume **not** $\phi^{\sim}$. For the purpose of comparison, we will not enforce coherence in this section when giving the definitions of these semantics. As we will see (cf. Subsection 6.2), coherence can be expressed by constraint clauses such as **not** $\phi^{\sim} \leftarrow \phi$. In particular, the enforcement of this principle is not a cause of intractability in these semantics.

Using the terminologies developed in this paper, we define

Definition 5.1 An assumption set W is said to be *consistent* with a program P iff $P \cup W \cup \{\bot \leftarrow \varphi, \mathbf{not}\,\varphi \mid \varphi \in H_L\} \not\vdash_d \bot$. $\Box$

Definition 5.2 Let P be a program and S be an assumption set. An assumption **not** ϕ is *weakly acceptable*[5] w.r.t. S iff **not** $\phi \in \mathcal{N}_P(S)$.

An assumption set W is *admissible* w.r.t. S iff it is consistent with P, and for each **not** $\phi \in W$, **not** ϕ is weakly acceptable w.r.t. S.

The *ideal skeptical semantics (ISS)* is defined by the greatest admissible set satisfying: for any other admissible set W', $W \cup W'$ is again admissible. $\Box$

ISS can be viewed as an instance of the *optimal fixpoint* theory, studied earlier by Manna and Shamir [20, 21]. It can be shown that for normal programs, ISS is an optimal fixpoint of the operator $\mathcal{F}_P^2$ (cf. Subsection 4.2 and also [33] for properties of $\mathcal{F}_P^2$). It is known that the optimal fixpoint uniquely exists but is difficult to compute [22].

[5] It is called *acceptable* in [1]. This terminology have been used earlier in this paper to refer to a different concept.

Definition 5.3 Let P be a program and ISS be its ideal skeptical semantics. Define a function $\mathcal{ADP}_P$ over sets S of assumptions:

$$\mathcal{ADP}_P(S) = \mathcal{N}_P(S) \cap ISS$$

The *well-founded semantics for extended programs (WFS0)* is defined by the least fixpoint of the operator $\mathcal{ADP}_P$. $\Box$

Theorem 5.1 *Given a program P, it is NP-hard to determine if an assumption is in its WFS0.*

Proof: For the reduction used in Theorem 3.3, simply add into P an additional program clause

$$\bot \leftarrow \phi, \mathbf{not}\ d$$

where d is a new symbol. We show that $\mathbf{not}\ d$ is in WFS0 iff (U, C) is unsatisfiable.

First note that (i) any assumption $\mathbf{not}\ \varphi \in \mathbf{not}\ (P)$ is trivially weakly acceptable, since φ does not appear in the head of any clause; (ii) $\mathbf{not}\ (P)$ is the least fixpoint of $\mathcal{N}_P$; and (iii) the ISS of P coincides with its WFS0. Thus, its WFS0 is equivalent to the intersection of all maximal consistent subsets of $\mathbf{not}\ (P)$. In addition, since $\{\mathbf{not}\ d\}$ is consistent with P, it is in at least one maximal consistent subset of $\mathbf{not}\ (P)$.

Now suppose $\mathbf{not}\ d$ is in WFS0 and we show that (U, C) is unsatisfiable. Suppose this is not true, i.e., (U, C) is satisfiable. Then it has a model, say M. Let S_M be the set of assumptions corresponding to M. Note that S_M is a maximal consistent subset of $\mathbf{not}\ (P)$. It follows that $P \cup S_M \vdash_d \phi$ and consequently $P \cup S_M \cup \{\mathbf{not}\ d\} \vdash_d \bot$. Hence, $\mathbf{not}\ d \notin S_M$. This contradicts the assumption that $\mathbf{not}\ d$ is in WFS0 and therefore in every maximal consistent subset of $\mathbf{not}\ (P)$.

Now assume (U, C) is unsatisfiable and show $\mathbf{not}\ d$ is in WFS0. Clearly, that (U, C) does not have a model implies that for any consistent assumption set S, $P \cup S \not\vdash_d \phi$. Thus, $\mathbf{not}\ d$ is in every maximal consistent subset of $\mathbf{not}\ (P)$, and hence in WFS0. $\Box$

Corollary 5.2 Given a program P, it is NP-hard to determine if an assumption is in its ISS. $\Box$

The same reduction can in fact be used to show that any semantics that is based on the intersection of all maximal consistent subsets is intractable. Note that preserving assumptions maximally is the same as removing assumptions minimally. Thus, any skeptical semantics that is based on removing assumptions as few as possible is intractable. This explains the precise difference between CGS and WFS0.

Example 5.1 Let

$$P = \{\bot \leftarrow \mathbf{not}\, a;\ \bot \leftarrow \mathbf{not}\, a, \mathbf{not}\, b\}.$$

With respect to **not** (P), its WFS0 is $\{\mathbf{not}\, b\}$ but its CGS is the empty set. $\square$

¿From this example, we can easily see that CGS is not minimal in that it may "over revise" a generated set of assumptions by removing more assumptions than necessary. However, non-minimal revision is often useful in practice. For example, in repairing a broken system of components, it is often more cost-effective to replace a sufficient subset of the components rather than spend the extra time to find the minimal subset–particularly if the behavior of the components is interdependent. For applications where minimal revision is desirable, tractable semantics can be used as a mechanism for efficiently generating an upper-bound approximation to the set of assumptions to be removed. It is an open question how to refine revisions in the direction of minimality while still maintaining tractability. The more informative grounded semantics presented in Subsection 6.1 represents a first attempt in this direction.

We now show that CGS can serve as an upper-bound approximation to WFS0.

Theorem 5.3 *Let P be a program, W be the set of assumptions corresponding to WFS0 and W' the set of assumptions corresponding to CGS. Then $W' \subseteq W$.*

Proof: By an induction on the sequence of constructing the least fixpoint: $S_0 = \emptyset$, and $S_{i+1} = \mathcal{A}_P(S_i)$, for all $i \geq 1$. (Recall that CGS is defined by the least fixpoint of the operator $\mathcal{A}_P(S) = \mathcal{N}_P(S) \cap \mathcal{R}_P(S)$.) The base case is trivial. For the inductive case, assume $\mathcal{A}_P(S_i) \subseteq ISS$ and show $\mathcal{A}_P(S_{i+1}) \subseteq ISS$. In this case, for each $\mathbf{not}\, \varphi$ in $\mathcal{A}_P(S_{i+1})$ but not in $\mathcal{A}_P(S_i)$, by the definition of $\mathcal{N}_P$, we know $\mathbf{not}\, \varphi$ is weakly acceptable w.r.t. S_i. By the induction hypothesis, we know $S_i \subseteq ISS$, and hence $\mathbf{not}\, \varphi$ is weakly acceptable w.r.t. ISS. We thus only need to show that $\{\mathbf{not}\, \varphi\} \cup ISS$ is consistent with P. Suppose this is not true. Then we have

$$P \cup ISS \cup \{\mathbf{not}\, \varphi\} \cup \{\bot \leftarrow \xi, \mathbf{not}\, \xi \mid \xi \in H_L\} \vdash_d \bot\ .$$

This implies that $\mathbf{not}\, \varphi$ is not in every maximal weakly acceptable set (known as *preferred extension* [8]). This contradicts to the well-known result that every assumption in the well-founded semantics is contained in every preferred extension [8]. $\square$

Dung's grounded semantics in general is different from WFS0. For example, with the following program

$$P = \{a \leftarrow \mathbf{not}\, d;\ d \leftarrow \mathbf{not}\, c;\ d^{\sim} \leftarrow \mathbf{not}\, c\}$$

the grounded semantics is $\{\textbf{not}\, d\}$ while both CGS and WFS0 are the empty set. Despite the difference, our result shows that these two semantics have an important common subset.

6 An Iterative Framework for Belief Revision

In this section we describe a general framework for iterative, nonmonotonic belief revision and present two applications of it. In this framework the intended semantics is determined by the least fixpoint of an operator, which we call *an assumption operator.*

Definition 6.1 Let P be a program and S be an assumption set. An *assumption operator* Φ_P is a unary function over sets of assumptions, generally defined as:

$$\Phi_P(S) = \mathcal{R}_k(S) \cap ... \cap \mathcal{R}_1(S) \cap \mathcal{N}(S)$$

where $k \geq 0$, $\mathcal{R}_i$'s and $\mathcal{N}$ are unary functions over sets of assumptions.[6] □

The intended use of this definition is as follows: the function $\mathcal{N}$, which may be composed of other functions, generates assumptions which are then revised (in fact constrained by the intersection) by the *revision operators* $\mathcal{R}_i$.

Different operators may yield different semantics. A semantics of this class is guaranteed to be tractable if the underlying assumption operator can be computed tractably. More precisely, if computing the assumption operator takes $O(\mathcal{P})$ time, where $\mathcal{P}$ is a polynomial, then computing the resulting semantics is bounded by $O(n \times \mathcal{P})$.

As a special case, when $k = 0$ no revision is specified. This is the case for normal programs. In fact, it can be shown that if $\Phi_P = \mathcal{N}_P$, then the least fixpoint of Φ_P corresponds to the well-founded model (cf. [2, 14]).

That an operator Φ_P is monotonic is essential to guarantee the existence of the least fixpoint. However, this can be determined by each participating function being monotonic.

Theorem 6.1 *Let $\Phi_P(S) = \mathcal{R}_k(S) \cap ... \cap \mathcal{R}_1(S) \cap \mathcal{N}(S)$. Φ_P is monotonic if each of the participating functions is monotonic.*
Proof: By an easy induction on the number of participating functions. □

The existence of the least fixpoint of Φ_P is independent of whether a program is contradictory or not. Thus even a contradictory program is defined semantically. However, we also want to know when a consistent semantics is guaranteed. This leads to the notion of sound revision operators.

[6]Recall that the same operator can be expressed by a function composition: $\Phi_P(S) = \mathcal{R}'_k \circ ... \circ \mathcal{R}'_1 \circ \mathcal{N}(S)$. We will however focus on the former expression.

Definition 6.2 Let P be a program. A revision operator $\mathcal{R}$ is said to be *sound* w.r.t. an assumption generation operator $\mathcal{N}$ iff, for any assumption set S, whenever $P \cup S \not\vdash_d \bot$ we have $P \cup (\mathcal{R}(S) \cap \mathcal{N}(S)) \not\vdash_d \bot$. $\square$

Theorem 6.2 *Let P be a non-contradictory program such that $\Phi_P(S) = \mathcal{R}(S) \cap \mathcal{N}(S)$, where $\mathcal{R}$ and $\mathcal{N}$ are monotonic functions. Further assume $\mathcal{R}$ is a sound revision operator w.r.t. $\mathcal{N}$. Then for the least fixpoint of Φ_P, denoted $fix(\Phi_P)$, we have $P \cup fix(\Phi_P) \not\vdash_d \bot$.*
Proof: Since P is non-contradictory, we have $P \not\vdash_d \bot$. Using the fact that $\mathcal{R}$ is sound w.r.t. $\mathcal{N}$, it is easy to show by induction (or transfinite induction when Φ_P is not continuous) that $P \cup fix(\Phi_P) \not\vdash_d \bot$. $\square$

The reason that the grounded semantics (CGS as well) is consistent whenever possible can be seen as a result of a sound revision operator. It is easy to verify that the revision operator $\mathcal{DR}_P$ (cf. Definition 3.5) is sound w.r.t. $\mathcal{DN}_P$.

The framework proposed here can accommodate very general notions of acceptability, which may be useful for formalizing alternative attack relations, or for applications where there are multiple attack relations.

6.1 A more informative grounded semantics

Sometimes the grounded semantics appears to be too skeptical. For example, consider the following example that Dung uses to illustrate the argumentation framework.

Example 6.1 (Why Tweety does not fly [9])

$$
\begin{aligned}
& fly(x) \leftarrow bird(x), \mathbf{not}\, ab_b(x) \\
& fly^{\sim}(x) \leftarrow penguin(x), \mathbf{not}\, ab_p(x) \\
& bird(x) \leftarrow penguin(x) \\
& ab_b(x) \leftarrow penguin(x), \mathbf{not}\, ab_p(x) \\
& penguin(tweety) \leftarrow
\end{aligned}
$$

Although the very idea of the argumentation framework suggests that Tweety does not fly, since an abnormal penguin (ab_p) is more specific than an abnormal bird (ab_b), the grounded semantics yields the empty assumption set, and thus it has no conclusion over whether Tweety can fly or not. Dung shows that the credulous argumentation semantics can correctly capture the intended meaning of this program. However, it is known that even credulous semantics for normal programs are not tractable (cf. [6]). $\square$

It can be argued that in general RAA-attack is weaker than g-attack: any RAA-attack from B to $\mathbf{not}\,\phi$ should be neutralized by B being g-attacked by $S \cup \{\mathbf{not}\,\phi\}$. This leads to the following more tolerable revision operator

$$\mathcal{RS}_P(S) =$$
$$\{\mathbf{not}\,\phi \mid \text{ for any } B, B \ RAA\text{-attacks } \{\mathbf{not}\,\phi\} \Rightarrow S \cup \{\mathbf{not}\,\phi\} \text{ g-attacks } B\}$$

The following claims are easy to verify.

Lemma 6.1 (i) $\mathcal{RS}_P$ is monotonic.
(ii) $\mathcal{R}_P(S) \subseteq \mathcal{RS}_P(S)$. $\square$

Thus the semantics defined by the least fixpoint of the operator $\mathcal{AS}_P(S) = \mathcal{RS}_P(S) \cap \mathcal{N}_P(S)$ is well-defined. In addition, a semantics based on $\mathcal{RS}_P$ is more informative than a semantics based on $\mathcal{R}_P$. The following proposition guarantees that the semantics defined this way assign a consistent semantics for any non-contradictory program.

Proposition 6.3 *$\mathcal{RS}_P$ is a sound revision operation, w.r.t. $\mathcal{N}_P$.*

Proof: It suffices to show that given an assumption set S, $P \cup (\mathcal{N}_P(S) \cap \mathcal{RS}_P(S)) \not\vdash_d \perp$. For the sake of contradiction, assume it does not hold, i.e., $P \cup (\mathcal{N}_P(S) \cap \mathcal{RS}_P(S)) \vdash_d \perp$. Let $B = \mathcal{N}_P(S) \cap \mathcal{RS}_P(S)$. Consider any $\mathbf{not}\,\phi_0 \in B - S$. Clearly, $\mathbf{not}\,\phi_0$ is RAA-attacked by $B - \{\mathbf{not}\,\phi_0\}$. By the definition of $\mathcal{RS}_P$, we know there exists $\mathbf{not}\,\phi_1 \in B - S$ such that $B - \{\mathbf{not}\,\phi_0\}$ is g-attacked by $S \cup \{\mathbf{not}\,\phi_0\}$, i.e., $P \cup S \cup \{\mathbf{not}\,\phi_0\} \vdash_d \phi_1$. Continuing, since B is finite, we have

$$\begin{array}{l} P \cup S \cup \{\mathbf{not}\,\phi_0\} \vdash_d \phi_1 \\ \ldots\ldots \\ P \cup S \cup \{\mathbf{not}\,\phi_{n-1}\} \vdash_d \phi_n \\ P \cup S \cup \{\mathbf{not}\,\phi_n\} \vdash_d \phi_k \end{array}$$

where $\phi_k = \phi_i$, for some $0 \leq i \leq n$. That is, a loop is formed. This implies that at least $\mathbf{not}\,\phi_k \notin \mathcal{N}_P(S)$, which is contradictory to the assumption that $\mathbf{not}\,\phi_k \in B - S$. $\square$

As we have already shown that computing $\mathcal{N}_P$ is tractable (cf. Theorem 4.2), the tractability of this new semantics can be guaranteed by showing that computing $\mathcal{RS}_P$ is also tractable.

Proposition 6.4 *Given an assumption set S, the problem of computing $\mathcal{RS}_P(S)$ is tractable.*

Proof: We present a naive algorithm whose tractability and correctness are easily verifiable.

Recall that we showed in Theorem 4.4 that $\mathcal{S}_{P_S}(\perp)$ is exactly the set of assumptions that are not acceptable in the grounded semantics due to RAA-attacks. Since $\mathcal{R}_P(S) \subseteq \mathcal{RS}_P(S)$, we only need to consider, for each $\mathbf{not}\,\phi \in \mathcal{S}_{P_S}(\perp)$, whether $\mathbf{not}\,\phi \in \mathcal{RS}_P(S)$, i.e., if for each argument D that RAA-attacks $\mathbf{not}\,\phi$, $S \cup \{\mathbf{not}\,\phi\}$ g-attacks D, then $\mathbf{not}\,\phi \in \mathcal{RS}_P(S)$.

Let $\mathbf{not}\,\phi \in \mathcal{RS}_P(S)$. Say an assumption set D is *RAA-attacking against* $\mathbf{not}\,\phi$ if there exists a subprogram $P'_S \subseteq P_S$ such that $P'_S \cup \{\mathbf{not}\,\phi\} \cup D$ is minimal satisfying $P'_S \cup \{\mathbf{not}\,\phi\} \cup D \vdash_d \ \perp$ (in other words, D is the set of assumptions in the clauses in P_S that are necessarily involved in a derivation of $\perp$). Though enumerating all such D's may require exponential time, it takes linear time to obtain the union of all such D's (by a straightforward trace of the program). Denote this union by $U_{\mathbf{not}\,\phi}$. Let $N_{\mathbf{not}\,\phi} = \{\mathbf{not}\,\xi \in \mathcal{S}_{P_S}(\perp) \mid P \cup S \cup \{\mathbf{not}\,\phi\} \vdash_d \xi\}$. If $U_{\mathbf{not}\,\phi} - N_{\mathbf{not}\,\phi}$ no longer RAA-attacks $\mathbf{not}\,\phi$, then $\mathbf{not}\,\phi \in \mathcal{RS}_P$ since every RAA-attacking argument has been g-attacked by $S \cup \{\mathbf{not}\,\phi\}$; otherwise, $\mathbf{not}\,\phi \notin \mathcal{RS}_P$ since at least one RAA-attacking argument is not g-attacked by $S \cup \{\phi\}$. $\square$

This modification of the belief revision operator also solves the problem that sometimes Dung's original belief revision operator may remove an assumption unnecessarily. Consider

$$P = \{a \leftarrow \mathbf{not}\,b;\ b \leftarrow \mathbf{not}\,c;\ \perp\leftarrow a, b\}.$$

According to our formulation, with $S_0 = \emptyset$, the grounded semantics can be determined by first computing $\mathcal{N}_P(S_0) = \{\mathbf{not}\,c\}$ and $\mathcal{R}_P(S_0) = \emptyset$, and then taking the intersection. Hence, the grounded semantics is specified by the empty set of assumptions. ¿From the belief revision point of view, $\{\mathbf{not}\,c\}$ does not need to be revised at all, since it does not cause any trouble. It is removed simply because the assumption $\mathbf{not}\,b$, which is g-attacked by it, *RAA*-attacks it.

6.2 Extended logic programming extended

As another example of applying the general framework introduced in this paper, we consider a programming extension by allowing the head of a clause to be an assumption. That is, a clause may take the form $\theta \leftarrow \alpha, \beta$ where $\theta \in \mathbf{not}\,(H_L)$. We call a clause of this form an *assumption-deriving clause.* ¿From now on an extended program may include zero or more assumption-deriving clauses. For the obvious reason, we also assume a program P always contains the constraints $\{\perp\leftarrow \phi, \mathbf{not}\,\phi \mid \phi \in H_L\}$.

An immediate application of this form of extended programming is to specify the so called *coherence principle* as given in [23] which states that whenever we believe in ϕ, $\mathbf{not}\,\phi^{\sim}$ should be assumed true. This can be viewed as part of a user's program, or being "compiled" from an extended program (without assumption-deriving clauses) in order to satisfy the semantic property of coherence. In the latter case, one only needs to add to his program the set of clauses

$$\{\mathbf{not}\,\phi \leftarrow \phi^{\sim}, \mathbf{not}\,\phi^{\sim} \leftarrow \phi \mid \phi \in H_L\}.$$

In this extended logic programming framework, we consider a reasoning agent whose beliefs are obtained, in terms of assumptions, as the result of (i) default assumptions; (ii) derived assumptions; and (iii) revising these assumptions. This can be formulated by the following equation:

$$\Phi_P(S) = R_P(S) \cap H_P \circ N_P(S)$$

where $N_P(S)$ is a function that generates default assumptions, H_P adds derived assumptions, and R_P performs revision. Specifically, we define[7]

$$\begin{aligned} N_P(S) &= \mathcal{N}_P(S) \text{ (Definition 4.2)} \\ H_P(S) &= \{\mathbf{not}\,\phi \mid P \cup S \vdash_d \mathbf{not}\,\phi \text{ or } \mathbf{not}\,\phi \in S\} \\ R_P(S) &= \overline{\mathcal{Z}_{P_S}(\perp)} \end{aligned}$$

where $\mathcal{Z}_{P_S}(\perp)$ is an extension of the notion of *support* as given in Definition 4.5 to account also for derived assumptions.

Definition 6.3 Let P_S be a program obtained from program P by S.

The *support* of ϕ w.r.t. P_S, denoted $\mathcal{Z}_{P_S}(\phi)$ where ϕ is either an atom or an assumption, is defined inductively as:

$\mathcal{Z}_{P_S}(\phi)$ is the smallest set satisfying, for any clause $\phi \leftarrow \alpha, \beta \in P_S$, if $P_S \cup \mathbf{not}\,(P_S) \vdash_d \xi$, for all $\xi \in \alpha$, then $\beta \subseteq \mathcal{Z}_{P_S}(\phi)$, and inductively, $\mathcal{Z}_{P_S}(\xi) \subseteq \mathcal{Z}_{P_S}(\phi)$ for each $\xi \in \alpha$, and $\mathcal{Z}_{P_S}(\mathbf{not}\,\tau) \subseteq \mathcal{Z}_{P_S}(\phi)$, for each $\mathbf{not}\,\tau \in \beta$. $\Box$

Example 6.2 Let $P = \{a \leftarrow \mathbf{not}\,b;\ a^{\sim} \leftarrow \mathbf{not}\,c;\ \mathbf{not}\,b \leftarrow b^{\sim};\ b^{\sim} \leftarrow \mathbf{not}\,d\}$. With $S = \emptyset$, we have $\mathcal{Z}_{P_S}(\perp) = \{\mathbf{not}\,b, \mathbf{not}\,c, \mathbf{not}\,d\}$. $\Box$

It is easy to see that both R_P and H_P are monotonic. Thus the operator Φ_P is monotonic and the semantics based on its least fixpoint is well-defined. It can be shown that the operator R_P is also sound w.r.t. $\mathcal{N}_P$. Thus, the semantics is consistent for non-contradictory programs. In addition, it can be verified that computing this semantics is bounded by $O(n^2)$.

[7] Note that the derivation relation $\vdash_d$, and the program reduction (Definition 4.4) can be used directly for extended programs with assumption-deriving clauses.

7 Other Related Work

The semantics based on the notion of contradiction removal (e.g., CRS in [25] and CRSX in [27]) employ a method to identify some minimal sets of *base* assumptions. First, there are non-contradictory programs that are considered not revisable under CRSX, and thus their semantics are not defined in it. Second, minimal revision in CRSX is defined w.r.t. base assumptions. E.g., with $P = \{\perp \leftarrow a;\ a \leftarrow \mathbf{not}\, b;\ b \leftarrow \mathbf{not}\, c;\ c \leftarrow \mathbf{not}\, d\}$, while the unique CRSX model makes everything *undefined* (by removing $\mathbf{not}\, d$), the grounded semantics only "removes" $\mathbf{not}\, b$, and as the result we still have $\mathbf{not}\, d$ and believe in c.

A number of attack relations have been studied in [4], but no skeptical semantics is studied. It would be interesting to investigate how the interactions of these attack relations may be though of as belief revision.

8 Final Remarks

One of the main contributions of this paper is a reformulation of Dung's grounded semantics, which yields two results: (i) the notion of acceptability has a compelling interpretation in terms of belief revision, and (ii) there is a tractable grounded semantics, called the common grounded semantics. These results lead to a general iterative framework for nonmonotonic belief revision, where a semantics is characterized by an assumption generation function and a belief revision function. We have shown some specific examples of applying this framework to obtain other tractable semantics.

In particular, we have shown that, due to different reasons, the grounded semantics of Dung, as well as the well-founded semantics for extended programs proposed by Alferes et al., are intractable. However, the common grounded semantics is a common subset of both of these semantics. It thus can be considered as a tractable approximation to the other semantics.

Although in this paper we have focused on skeptical semantics, we note that each acceptability operator can be used to specify a credulous semantics by considering its maximal fixpoints (cf. [9]).

Our framework for iterative belief revision relies on the assumption that the function Φ_P is continuous and hence computable by finite approximations. The question remains open as how to deal with non-ground programs with an infinite domain of assumptions.

9 Appendix

Here, we give an example of the reduction used in the proof of Theorem 3.3. Consider an instance of 3-SAT problem (U, C) where

$$U = \{u, v, w, x\}$$
$$C = \{u \vee \neg v \vee w,\ \neg u \vee v \vee x\}$$

The program P reduced from (U, C) is:

$$c_1 \leftarrow \mathbf{not}\ u^{\sim}$$
$$c_1 \leftarrow \mathbf{not}\ v$$
$$c_1 \leftarrow \mathbf{not}\ w^{\sim}$$

$$c_2 \leftarrow \mathbf{not}\ u$$
$$c_2 \leftarrow \mathbf{not}\ v^{\sim}$$
$$c_2 \leftarrow \mathbf{not}\ x^{\sim}$$

$$\phi \leftarrow c_1, c_2$$

$$\perp \leftarrow \mathbf{not}\ u, \mathbf{not}\ u^{\sim}$$
$$\perp \leftarrow \mathbf{not}\ v, \mathbf{not}\ v^{\sim}$$
$$\perp \leftarrow \mathbf{not}\ w, \mathbf{not}\ w^{\sim}$$
$$\perp \leftarrow \mathbf{not}\ x, \mathbf{not}\ x^{\sim}$$

Every sound argument that derives ϕ corresponds to a model. For example, argument $B = \{\mathbf{not}\ u^{\sim}, \mathbf{not}\ v^{\sim}\}$ is sound and $P \cup B \vdash_d \phi$. The corresponding partial, consistent assignment I_B assigns u and v to *true*. By extending I_B to include an assignment to w and x arbitrarily, we get a model.

On the converse, for example, $M = \{u, v, w, \neg x\}$ is a model of (U, C). The corresponding argument $S_M = \{\mathbf{not}\ u^{\sim}, \mathbf{not}\ v^{\sim}, \mathbf{not}\ w^{\sim}, \mathbf{not}\ x\}$ is sound and $P \cup S_M \vdash_d \phi$.

References

[1] J. Alferes, P. Dung, and L. Pereira. Scenario semantics of extended logic programs. In *Proc. 2nd International Workshop on LPANR*, pages 334–348. MIT Press, 1993.

[2] C. Baral and V. Subrahmanian. Dualities between alternative semantics for logic programming and nonmonotonic reasoning. In *Proc. First International Workshop on LPANR*, pages 69–86. MIT Press, 1991.

[3] H. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computing Science*, 68:135–154, 1989.

[4] A. Bondarenko, F. Toni, and R.A. Kowalski. An assumption-based framework for nonmonotonic reasoning. In *Proc. 2nd International Workshop on LPANR*, pages 171–189. MIT Press, 1993.

[5] J. Dix. A classification theory of semantics of normal logic programs: I. strong properties. *Foundamenta Informaticae*, 1994.

[6] J. Dix. Semantics of logic programs: their intuitions and formal properties (an overview). Technical report, University of Koblenz-Landau, 1995. Available from LPNMR Archives, ftp.ms.uky.edu/pub/lpnmr.

[7] J. Doyle. Reason maintenance and belief revision: foundations versus coherence theories. In Gardenfors, editor, *Belief Revision*, pages 29–51. Cambridge University Press, 1992.

[8] P. Dung. Negations as hypotheses: An abductive foundation for logic programming. In *Proc. 8th International Conference on Logic Programming*, pages 3–17, 1991.

[9] P. Dung. An argumentation semantics for logic programming with explicit negation. In *Proc. 10th ICLP*, pages 615–630, 1993.

[10] P. Dung. An argumentation theoretic foundation for logic programming. *J. Logic Programming*, 22:151–177, 1995.

[11] P. Dung. On the acceptability of argument and its fundamental rule in non-monotonic reasoning and logic programming and n-person game. *Artificial Intelligence*, 77:31–357, 1995. An extended abstract appeared in *Proc. IJCAI'93.*

[12] K. Eshghi and R.A. Kowalski. Abduction compared with negation by failure. In *Proc. 6th ICLP*, pages 234–254. MIT Press, 1989.

[13] P. Gardenfors, editor. *Belief Revision.* Cambridge University Press, 1992.

[14] A. Van Gelder. The alternating fixpoint of logic programs with negation. *J. Computer and System Sciences*, pages 185–221, 1993. The preliminary version appears in the *Proc. 8th ACM PODS*, pages 1-10,1989.

[15] A. Van Gelder, K. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 23(4):620–650, 1990.

[16] M. Gelfond and V. Lifschitz. Logical programs with classical negation. In *Proc. 7th ICLP*, pages 579–597. MIT Press, 1990.

[17] M. Gelfond and V. Lifschitz. Representing actions and changes by logic programs. *J. Logic Programming*, 17:301–322, 1994.

[18] A. Kakas, P. Mancarella, and P.M. Dung. The acceptability semantics for logic programs. In *Proc. 11th ICLP*, pages 504–519, 1994.

[19] R.A. Kowalski and F. Dadri. Logic programs with exceptions. In *Proc. 7th ICLP*, pages 490–504. MIT Press, 1990.

[20] Z. Manna and A. Shamir. The theoretical aspects of the optimal fixedpoint. *SIAM J. Computing*, 5(3):414–426, September 1976.

[21] Z. Manna and A. Shamir. The optimal approach to recursive programs. *Communications of the ACM*, 20(11):824–831, November 1977.

[22] Z. Manna and A. Shamir. The convergence of functions ot fixpoint of recursive definitions. *Theoretical Computer Science*, 6:109–141, 1978.

[23] L. Pereira and J. Alferes. Well founded semantics for logic programs with explicit negation. In *Proc. 10th ECAI*, pages 102–106, 1992.

[24] L. Pereira and J. Alferes. Optative reasoning with scenario semantics. In *Proc. 10th ICLP*, pages 601–615, 1993.

[25] L. Pereira, J. Alferes, and J. Aparício. Contradiction removal within well-founded semantics. In *Proc. First International Workshop on LPANR*, pages 105–119, 1991.

[26] L. Pereira, J. Alferes, and J. Aparício. Counterfactual reasoning based on revising assumptions. In *Proc. ILPS '91*, pages 566–580. MIT Press, 1991.

[27] L. Pereira, J. Aparício, and J. Alferes. Non-monotonic reasoning with logic programming. *J. Logic Programming*, 17:227–263, 1993.

[28] L. Pereira, C. Damásio, and J. Alferes. Diagnosis and debugging as contradiction removal. In *Proc. 2nd International Workshop on LPANR*, pages 316–332. MIT Press, 1993.

[29] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Mathematics*, 5(2):285–309, 1955.

[30] C. Witteveen and G. Brewka. Skeptical reason maintenance and belief revision. *Artificial Intelligence*, 61:1–36, 1993.

[31] C. Witteveen and C. M. Jonker. Revision by expansion in logic programs. Technical Report Report 93-02, Faculty of Mathematics and Computer Science, Delft University of Technology, 1993.

[32] J. You and R. Cartwright. Tractable argumentation semantics via iterative belief revision. In *Proc. ILPS '94*, pages 239–254. MIT Press, 1994.

[33] J. You and L. Yuan. On the equivalence of semantics for normal logic programs. *J. Logic Programming*, 22(3):209–219, March 1995.