Revision by Communication:

Program Revision by Consulting Weaker Semantics

C. Witteveen and W. van der Hoek

UU-CS-1994-56
December 1994

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Revision by Communication:

Program Revision by Consulting Weaker Semantics

C. Witteveen and W. van der Hoek

Technical Report UU-CS-1994-56
December 1994

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924-3275

Revision by Communication

program revision by consulting weaker semantics

Cees Witteveen* Wiebe van der Hoek†

Abstract

We deal with the problem of revising logic programs that, according to some non-monotonic semantics, do not have an acceptable model. We propose to study such revisions in a framework where a number of semantical agents is distinguished, each associated with different semantics but interpreting the same program. It is well-known that for logic programs the different semantics can be partially ordered by inferential strength. If an agent cannot find an acceptable model for the program, he has to perform program revision. In this revision process, the agent may consult his weaker colleagues, adds the information they can infer to the program and tries to find an acceptable model for the expanded program. In this paper we will concentrate on the kind of information needed to find successful revisions of programs. This framework is a strict refinement and generalization of existing work in program revision. We point out some parameters along which our framework can be analyzed.

Keywords: Logic Programming, Revision, Non-monotonic semantics.

1 Introduction

In the logic programming community one usually distinguishes the set of *intended* or *acceptable* models of a program from its set of *classical* models, the former usually

*Delft University of Technology, Dept of Mathematics and Computer Science P.O.Box 356, 2600 AJ Delft, The Netherlands, witt@cs.tudelft.nl

†Utrecht University, Dept of Computer Science, Utrecht, The Netherlands, wiebe@cs.ruu.nl

being a strict subset of the latter.

By using a set of intended models, one can obtain stronger (but defeasible) conclusions from a program than by using classical models. This increase in inferential power, however, does not hold for all programs: sometimes agents cannot establish an intended model for a program, even if the program is classically consistent. In this case there is a need for program revision: the current program does not satisfy any interpretation acceptable to the agent.

For classical theories, there is a nice framework ([4]) for doing revision that essentially is *inconsistency-driven*. Briefly, it comes down to narrowing the theory in order to remove inconsistencies by giving up some existing piece of information.

In this paper we will not deal with classically inconsistent theories. Instead, we will investigate classically consistent theories, which under a given non-monotonic semantics S, do not have an acceptable model. Let us call such a theory *non-monotonically inconsistent* or *nm-inconsistent*. Solving nm-inconsistency is quite different from solving the classical inconsistency problem:

- nm-inconsistent theories have a classical model and therefore retraction of statements is not a prerequisite to restore consistency.

A number of attempts like [6, 11, 18, 17] recognized this difference and established a framework for revision of nm-inconsistent programs, not by *retracting* information contained in the program, but by *adding* information to it in order to restore nm-consistency. In these approaches, however, the existence of different, but related, non-monotonic semantics for programs has not been recognized: here, we propose to exploit the following observation:

- in general there are several related non-monotonic semantics for a given class of theories. At least for logic programs these semantics can be *partially* ordered according to inferential strength.

In existing approaches of solving inconsistency, the only source of additional information was consulting *classical* models to add information to the program. Moreover, almost all attention has been given to the revision of programs under the stable model semantics. In this paper we will extend the existing framework for program revision as given in [6, 11, 18, 17] by paying attention to the interrelations

between different semantics of programs.

After we have given some preliminaries in 1.1, we will present our general framework for revision by communication in Section 2. In Section 3 we will concentrate on one aspect of this framework: the role that different kinds of information play in revising programs. Then, in Section 4 we give some results that can be easily obtained about the framework and that indicate some of its strength and limitations.

1.1 Preliminaries

We assume the reader to be familiar with some basic terminology used in logic programming, as for example in Lloyd [8]. A *normal program rule* r is a directed propositional clause of the form $A \leftarrow B_1, \ldots, B_m, \neg D_1, \ldots, \neg D_n$, where $A, B_1, \ldots, D_1, \ldots, D_n$ are all atomic propositions. The head of such a rule r is denoted by $hd(r)$ and its body by $body(r) = \alpha(r) \wedge \beta(r)$, where $\alpha(r)$ denotes the conjunction of the B_i atoms and $\beta(r)$ denotes the conjunction of the negative atoms $\neg D_j$. A program is called *positive* if for every rule r, $\beta(r) = \top$ (the empty conjunction). *Constraints* are special rules of the form $\bot \leftarrow \alpha$, expressing that the conjunction of the literals occurring in α cannot be true. We do not allow $\bot$ (*falsum*) to occur in the body of any rule r. If l is a literal, $\bar{l}$ equals $\neg l$ if l is a positive atom and equals s if $l = \neg s$ for some $s \in B_P$. Likewise, $\neg L = \{\bar{l} \mid l \in L\}$.

As usual, interpretations and models of a program are denoted by maximal consistent subsets I of literals over the Herbrand base B_P of P, i.e., for every atom $a \in B_P$, either a or $\neg a$ occurs in an interpretation I. A *model* of a program P is an interpretation I satisfying all the rules and constraints of P. Here, a rule r is said to be satisfied by an interpretation I iff $I \models body(r)$ implies $I \models hd(r)$ and a constraint $\bot \leftarrow \alpha, \beta$ is satisfied by I iff $I \not\models \alpha \wedge \beta$ or, equivalently, I is a model of the constraint as a rule and I^+ does not contain the special atom $\bot$.

We denote the set of classical models of P by $Mod(P)$. The definitions of *minimal* model, *supported* model and *stable* model are assumed to be well-known. If M is both supported and minimal, M is called *positivistic* (see [1]).

To discuss properties of stable models, we use the Gelfond-Lifschitz reduction $G(P, M)$ of a program P with respect to a model M, where $G(P, M) = \{c \leftarrow \alpha \mid c \leftarrow \alpha \wedge \beta \in P,\ M \models \beta\}$

We use $Mod(P), MinMod(P), Supp(P), Pos(P), Stable(P)$ to denote the class of classical models, minimal models, supported models, positivistic models and stable models, respectively.

It is well-known that the following inclusion relations hold (cf [1]): For every logic program P, $Stable(P) \subseteq Pos(P) \subseteq Supp(P) \subseteq Mod(P)$ and $Pos(P) \subseteq MinMod(P) \subseteq Mod(P)$

In the subsequent sections, we will also need clause-representations of rules. If r is the rule $c \leftarrow a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n$, then the clause $C(r)$ *associated with* r is $C(r) = \{c, \neg a_1, \ldots,, \neg a_m, b_1, \ldots, b_n\}$. If P is a logic program, $C(P) = \{C(r) \mid r \in P\}$ is the set of clauses associated with P. Conversely, if $\mathcal{C}$ is a set of clauses, $Normal(\mathcal{C})$ refers to the set of *normal* logic programs P such that $\mathcal{C} = C(P)$. Note that, since we include constraints, for every set of clauses $\mathcal{C}$, there exists such a normal program P.

A clause C is called *positive* if it contains only positive atoms. A clause C is said to be a *prime implicate* of a set of clauses Σ iff $\Sigma \models C$ and there is no proper subset $C' \subset C$ such that $\Sigma \models C'$.

Let us finally mention some results that will be used in subsequent sections[1].

Lemma 1.1 (Marek & Truszczyński [10])
Let M be a model of a program P. Then $T_{G(P,M)} \uparrow \omega \subseteq M$.

Lemma 1.2 *Let P be a program and $M_{neg} = \{\neg a \mid a \in B_P\}$ an interpretation of P. Then the following statements are equivalent:*
(i) M_{neg} is a model of P.
(ii) M_{neg} is a stable model of P.
(iii) the body of every rule r of P contains at least one positive literal.

Lemma 1.3 (Expansion Lemma)
Let P be a program, $P' \subset P$ and M a model of P. Then, if M is a (minimal, supported, positivistic, stable) model of P', M also is a (minimal, supported, positivistic, stable) model of P.

PROOF Let M be a minimal model of P' and a model of P. Then it is immediate that M is also a minimal model of P. For supported models the lemma is also

[1]Short proofs will occur in the final version but are omitted here.

trivially true, since $M \in Sup(P')$ and $M \in Mod(P)$ implies that for all $a \in M$ there is at least one rule $r \in P' \subseteq P$ with $hd(r) = a$, such that $M \models hd(r) \wedge body(r)$. Therefore, M is also a supported model for P. Combining the results for supported an minimal models is sufficient to prove the property for positivistic models.

For stable models, assume that M is a stable model of P' and a model of P. Then $M^+ = lfpT_{G(P',M)} \subseteq lfp(T_{G(P,M)})$, where the last inclusion holds since $G(P', M)$ and $G(P, M)$ are positive programs and $G(P', M) \subseteq G(P, M)$,

Since M is a model of P, by Lemma 1.1, it follows that $lfp(T_{G(P,M)}) \subseteq M^+$. So $M^+ = lfp(T_{G(P,M)})$, and therefore $M \in Stable(P)$. □

2 Revision by Communication

What is considered as the intended meaning of a program clearly depends on the possible applications the program designer had in mind ([2]). It is generally agreed upon that these various intentions and notions of acceptability can be captured by *non-monotonic* interpretations of logic programs as, for example, the minimal model, supported model, positivistic model and stable model semantics. By using intended models instead of the full set of classical models, an agent is able to derive stronger (but defeasible) conclusions from a program than a classical agent is able to derive.

The increase in inferential power, however, does not hold for all programs: sometimes we cannot establish an intended model for a program, even if the program is classically consistent. In this case, there is a need for *program revision*. There exists a well-known framework for doing revision in classical theories. This kind of theory revision is *inconsistency*-driven ([13]) and revision is performed by *retracting* parts of the theory in order to obtain a consistent subtheory. Program revision using a non-monotonic logic, however, is not inconsistency-driven, since the program may have classical models, but we cannot find acceptable ones. This means that we have to change our idea about what should be the intended model of the program. Certainly, retraction of information is not the only way to solve this problem.

In [17] we showed that for logic programs, an interesting result could be obtained: we could provide every consistent logic program P with a stable model by adding a subset of its *classical* consequences to P.

As a matter of fact, this is a result typical for the existing framework: only information obtained from classical interpretations of programs is used in program revision. For logic programming, however, there exist several, related, non-monotonic semantics and they can be partially ordered according to their inferential strength. Since classical logic is the weakest among these semantics, this means that in the current framework we only used the weakest information possible to revise our program. Therefore, in this paper we will generalize the existing framework in a significant way. Instead of using one particular non-monotonic semantics and the use of classical information to perform program revision, we propose to do program revision in a larger, *distributed*, framework.

In this set-up, several agents are associated with the different (non-monotonic) semantics and each agent interprets the same program. These agents are partially ordered according to their inferential strength. As soon as an agent cannot find an intended model for the program, he is allowed to consult one or more weaker agents in parallel. The information he obtains from them will be compared and the results can be added to the program. Then the agent may try to find an intended model for the *expanded* program. In this way the existing revision framework becomes a set of borderline cases in the new framework.

To be more precise, let us assume that we have a set of agents each associated with a particular semantics $Sem \in \{Stable, Pos, Supp, MinMod, Mod\}$ and interpreting the same program P in their own way. If A is an agent, Sem_A denotes the semantics A uses in selecting the intended model of a program. This set of intended models is denoted by $Sem_A(P)$ and $Sem_A \models x$ denotes that x is true in all intended models $Sem_A(P)$.

Then we say that an agent A is weaker than an agent B, denoted by $A < B$ iff for every program P, $Sem_B(P) \subseteq Sem_A(P)$ and for at least one program P a strict inclusion holds.

The idea of revision by communication essentially is that if an agent Sem cannot find an acceptable model for a program P, he may consult a weaker colleague Sem'. This agent can tell him what he considers as true in his set of intended models. The agent Sem can add this information to the program and tries to find an acceptable model for the expanded program. So the intended models of P are equal to

$$Sem(P + \{x \in R \mid Sem'(P) \models x\})$$

where R is a set of statements that may be communicated to Sem, e.g. some subsets of atoms or rules.

The framework sketched in [17] in fact was a very restricted one it was confined to stable agents using classical consequences only. The intended models of the revised theory were obtained as the stable model of classical expansions:

$$Stable(P + \{x \in R \mid Mod(P) \models x\})$$

In this paper, we will primarily concentrate on the nature of the information to be provided if an agent Sem consults a weaker colleague Sem'. In Section 4 we will briefly mention some general observation about this framework. Furthermore, we will discuss revision for *normal logic programs* and always assume that the programs to be revised are *classically consistent.*

3 Some General Results

Given some semantics $Sem > Sem'$ we would like to know which information Sem needs from Sem' in order to revise a program P successfully, i.e. obtaining an expansion P' such that $Sem(P') \neq \emptyset$. Since the information delivered by Sem' has to be added to the program, we distinguish between atomic or factual information and rule information provided by Sem'.

It is not difficult to show that in general, atomic information is not sufficient to obtain an acceptable model for an arbitrary agent[2].

Lemma 3.1 *There are programs P and semantics $Sem < Sem'$ such that*

1. $Sem'(P) \neq \emptyset$,
2. $Sem(P + F) = \emptyset$ *for every* $F \subseteq \{a \in B_P \mid Sem'(P) \models a\}$

PROOF Consider the following program:

$$\begin{aligned} P: \quad a &\leftarrow \neg a, b \\ b &\leftarrow \neg c \\ c &\leftarrow \neg b \\ d &\leftarrow \neg d, c \end{aligned}$$

[2]Atomic information is sufficient if we do not require the skeptical form of revision, see Section 4

Take $Sem = Stable$ or $Sem = Pos$ and $Sem' = MinMod$. P has two minimal models: $M_1 = \{a, b\}$ and $M_2 = \{c, d\}$ and no positivistic or stable model. Note, however that every positivistic or stable agent ends up with an expansion $P = P'$, hence there exists no expansion for him having a positivistic or stable model. □

3.1 Rule information

Since atomic information does not always help, we will try to establish results for agents also accepting rule information from their weaker colleagues.

Definition 3.2 If P is a program

- $Rule(B_P)$ denotes the set of all *normal rules* that can be constructed from atoms in B_P.
- $Contra(P) \subseteq Rule(B_P)$ denotes the set of all logical contrapositives $Contra(r)$ of rules r such that $Contra(r)$ again is a normal rule. So for example, if $a \leftarrow \neg b, c$ is a rule $Contra(r) = \{b \leftarrow \neg a, c\ ,\ \bot \leftarrow \neg a, \neg b, c\}$

Clearly, $Contra(P) \subseteq Rule(B_P)$.

Remark. There is a close correspondence between shift operations as defined in [2] and taking contrapositives. In fact, contrapositives can be considered as bi-directional shift operations where literals can be moved from the head of a rule to the body and vice-versa. ■

Before looking at the general case we will try to establish the information content of contrapositives of rules. Contrapositives have been used to give a logical reconstruction of the idea of *dependency-directed backtracking* (see [3]), used in both logic programming and truth maintenance as a revision technique. In [15] we have shown that in a three-valued logic, adding contrapositives is successful in obtaining a stable model of the expanded program. For 2-valued semantics, we will show that this picture is different. Note that $P \models Contra(P)$, so we can concentrate on classical (or minimal) reasoners as our information source.

Our first result is that even for positivistic reasoners it is sufficient to use contrapositive information:

Lemma 3.3 *Let P be a program. Then:*

$$Pos(P + \{x \in Contra(P) | Mod(P) \models x\}) \neq \emptyset$$

PROOF It is sufficient to show that there exists a set $F \subseteq Contra(P)$ such that $P + F$ has a positivistic model. For, let M_{pos} be such a positivistic model. Since $Mod(P) \models Contra(P)$, clearly, $M_{pos} \models Contra(P) - F$ and, by the Expansion Lemma, M is a positivistic model of $P + Contra(P)$.

Since P is assumed to be consistent, P has at least one minimal model M. Recall that $C(P)$ is the set of clauses associated with P. We distinguish the following cases:

1. M does not contain a positive literal.
   By Lemma 1.2, M is a stable model of P, and since $Stable(P) \subseteq Supp(P)$, M is a supported model of P.

2. M contains at least one positive literal.
   Let $M^+ = \{c_1, c_2, \ldots, c_m\}$. Since M is a minimal model, for every $c_i \in M^+$ there is at least one (not necessarily positive) clause $C_i = \{c_i, l_1, \ldots, l_m\}$. in $C(P)$ containing the atom c_i, such that M minimally satisfies C_i by making c_i true and making the remaining literals l_j, $j = 1, 2, \ldots, m$ false. For else, c_i could be safely removed from M^+, contradicting the fact that M is minimal. Then $M \models c_i$ and $M \models \neg l_j$ for $j = 1, \ldots m$, hence c_i is supported by the following rule $r(c_i)$ associated with such a clause C_i:

   $$r(c_i) = c_i \leftarrow \bar{l}_1, \bar{l}_2, \ldots, \bar{l}_m.$$

   Let $P_{M^+} = \{r(c_i) \mid c_i \in M^+\}$. By definition, M is a supported model of P_{M^+}.

   Since M is a minimal model of $P \subset P'$, M is also a minimal model of P'. Hence M is a positivistic model of P'.

   For the remaining clauses $C(r) \in C(P)$, take the original rules $r \in P$ and add them to P_{M^+}. Note that since $M \models C(P)$, $M \models r$ for every such a rule r.

   Let the resulting program be P' and note that $P' - P \subseteq Contra(P)$. By the Expansion Lemma (Lemma 1.3), M is a positivistic model of P'.

□

In the case of the stable model semantics, adding contrapositives in general is *not* sufficient. We present a simple example:

Example 3.4 Take the following program:

$$\begin{aligned} P: \quad b &\leftarrow \neg a \\ a &\leftarrow b \\ b &\leftarrow a \end{aligned}$$

The unique (classical) model of P is $M = \{a, b\}$. Since $Mod(P) = Mod(P + F)$ for every $F \subseteq Contra(P)$, if there is a stable model of some $P + F$, it should be equal to M.
Note, however, that for each such a program $P' = P + F$, we have

$$G(P', M) \subseteq \{a \leftarrow b, b \leftarrow a\}$$

Hence, $lfp(T_{G(P',M)}) = \emptyset \neq M^+$ and M cannot be stable.

To summarize the results for adding contrapositives, we state the following theorem as an easy consequence of both Lemma 3.3 and the previous example:

Theorem 3.5 *For every program P and semantics Sem such that $Stable > Sem$, $Sem(P + Contra(P)) \neq \emptyset$.*

3.2 Adding contrapositives and condensations

Since adding contrapositives is not always sufficient to help stable agents, we will investigate the use of other rule information. We will now allow for the addition of contrapositives of program rules in which some literals may have been removed. Such *condensations*, however, of rules are only allowed if they appear as classical consequences of P.
We will then show that combining contrapositives and condensations is sufficient to allow a stable reasoner to find an acceptable model for every program. This immediately implies that such information is sufficient to provide every weaker reasoner with an intended model.

A condensation[3] then of a program P is a program in which every rule of P occurs, except for the removal of some literals in the head or body of the rule.
We need to introduce some preliminaries.

Definition 3.6 A set of clauses $\mathcal{C}'$ is said to be a *condensation* of a set of clauses $\mathcal{C}$ iff $\mathcal{C}'$ is an inclusion minimal set of clauses such that

1. for every clause $C \in \mathcal{C}$ there exists a clause $C' \in \mathcal{C}'$ such that $C' \subseteq C$ and
2. $Mod(\mathcal{C}) = Mod(\mathcal{C}')$.

Remember that if a program P' is related to P by adding contrapositives, then $C(P) = C(P')$.

Definition 3.7 We say that a program P' can be obtained from a program P by *contracondens* operations if $C(P')$ is a condensation of $C(P)$. $ContraCondens(P)$ denotes the set of rules that can be obtained from P by applying contracondens operations.

It is often useful to consider *extreme* condensations of clauses and programs:

Definition 3.8 A set of clauses $\mathcal{C}'$ is a *maximal condensation* (m.c.) of a set of clauses $\mathcal{C}$ iff

1. $\mathcal{C}'$ is a condensation of $\mathcal{C}$ and
2. $\mathcal{C}'$ is *maximally condensed*, i.e. for every condensation $\mathcal{C}''$ of $\mathcal{C}'$, $\mathcal{C}'' = \mathcal{C}'$.

Analogously, a program P is a *maximally* condensed program, or *mclp*, iff $C(P)$ is a maximally condensed set of clauses and $C'(P)$ is a maximal condensation of P if $C'(P)$ is a maximal condensation of $C(P)$.

The following easy result shows that maximal condensations of P can be obtained by using prime implicates of $C(P)$:

Proposition 3.9
Let $\Pi(P)$ denote an inclusion-minimal set of prime-implicates of $C(P)$ such that for every $C \in C(P)$ there is a prime implicate C' of $C(P)$ with

[3]This notion is related to the notion of a condensation of a clause as discussed in [7] but not identical. In this latter article, a condensation relation is a relation between two clauses instead of sets of clauses.

- $C' \subseteq C$ *and*
- $C' \in \Pi(P)$.

Then $\Pi(P)$ *is a maximal condensation of* $C(P)$.

A nice property of maximal condensations we will need in the next section is that in every positive clause C of $\Pi(P)$ every atom $c \in C$ occurs in at least one minimal model M of P that also minimally satisfies C:

Observation 3.10
For every positive clause C *in* $\Pi(P)$ *and for every atom* c *in* C *there exists at least one minimal model* M *of* P *such that* M *minimally satisfies* C *and* M *makes* c *true.*

This property will be used in the next section to show that given an arbitrary consistent program P, we can construct a normal program P' from $\Pi(P)$ such that (i) $P' \subseteq ContraCondens(P)$ and (ii) $Stable(P') \neq \emptyset$.

3.3 Constructing stable models for mclp's

In this section we will show that condensing + shifting is successful for every class of programs with respect to the stable semantics.
The construction idea we will use can be summarized as follows:

1. We assume to be given a maximal condensation $\Pi = \Pi(P)$ of an arbitrary consistent program P.
2. From $\Pi(P)$ we construct a normal program P' and a stable model M'.
3. We show that $C(P') = \Pi(P)$, i.e. the model-theoretical interpretation of P' and P is identical.

Since for every logic program P there exists a maximal condensation $\Pi(P)$ of P, this shows that for every program P there exists a program $P + Contra(P)$ such that P' has a stable model.

As an important corollary, we note that shift operations are successful for the class of maximally condensed logic programs.

The method we will present constructs P' and M' in a finite number of stages $i = 0, 1, \ldots, n$. At every stage i, $i \geq 0$, the currently partial realization of P' is denoted as P^i and the (partial) model associated with P^i as M^i.
We will show that:

Claim 1 At every stage i, M^i will be a stable model of P^i and a partial model of Π, i.e. there exists at least one complete extension M of M^i which is a model of Π.

Claim 2 After a finite number of stages, P^i will be a subset of $Contra(\Pi)$ and M' will be a stable model of P'.

The stages themselves are defined as follows:
At Stage 0, let $P^0 = \emptyset$ and $M^0 = \emptyset$. Since M^0 is a stable model of P^0 and M^0 is extendible to a model M of Π, Claim 1 holds for Stage 0.
At Stage $i+1$, proceeding inductively, we have at our disposal a partial model M^i of Π, which is also a stable model of P^i, where $P^i \subseteq Contra(\Pi')$, for some subset Π' of Π.
Then we consider a set of clauses $\Pi^{i+1} = R(\Pi, M^i)$, derived from Π and M^i as follows:

1. remove every *clause* C' from Π containing a literal c such that $M^i \models c$,
2. remove in the remaining clauses C', every *literal* c such that $M^i \models \neg c$;
3. let the resulting set of clauses be Σ^{i+1} and construct a set Π^{i+1} of maximally condensed clauses from Σ^{i+1}.

Note that, since M^i has an extension M satisfying Π, Σ^{i+1} and Π^{i+1} must be satisfiable.
We distinguish two cases:

Case 1. Π^{i+1} does not contain any positive clause.

By Lemma 1.2, $M_{neg} = \neg(Atoms(\Pi) - Atoms(P^i))$, is a stable model of Π^{i+1}.

Note that $C(P^i) \subseteq \Pi$. Let $P'' \in Normal(\Pi - C(P^i))$ be an arbitrary normal program, $P^{i+1} = P^i \cup P''$ and $M^{i+1} = M^i \cup M_{neg}$.

Then it is not difficult to show that

(a) M^{i+1} is a stable model P^{i+1}

(b) $P^{i+1} \subseteq Contra(\Pi)$

This implies that Stage i is a final stage, $M' = M^{i+1}$ and $P' = P^{i+1}$, proving Claim 1+2.

Case 2. Π^{i+1} contains at least one positive clause $C'' = \{c_1, \ldots, c_k\}$.
Since Π^{i+1} is a maximally condensed set, by Observation 3.10, there is a minimal model M_m of Π^{i+1}, making c_1 true and every c_j, $j \geq 2$ false.
Let $C' \supseteq C''$ be a clause in Σ^{i+1} from which C'' has been derived and

$$C = \{c_1, \ldots, c_k, c_{k+1}, \ldots, c_m\}$$

be the clause in Π from which C' has been derived.

Consider the following rule $r(C)$ to be added to P^i to form the program P^{i+1}:

$$r(C) = c_1 \leftarrow \bar{c}_2, \ldots, \bar{c}_k, \bar{c}_{k+1}, \ldots, \bar{c}_m$$

Let $M^{i+1} = M^i \cup \{c_1, \bar{c}_2, \ldots, \bar{c}_m\}$. Now we have to prove that:

(a) M^{i+1} is a stable model of P^{i+1}.

(b) there is at least one extension M of M^{i+1} such that M is a model of Π.

These claims are easy to prove.

Finally, note that at the end of Stage $i+1$, either

$$C(P^{i+1}) = \Pi(P)$$

or

$$C(P^i) \subset C(P^{i+1}) \subset \Pi(P) \text{ and } C(P^{i+1}) - C(P^i) \neq \emptyset.$$

This means that after a finite number of stages, the construction of P' is completed.

¿From the properties of this construction, the following result can be easily derived:

Lemma 3.11

For every consistent maximally condensed set of clauses Π, *there is a normal program* $P' \subseteq Contra(\Pi)$, *such that* $Stable(P') \neq \emptyset$.

The procedure given in Figure 3.3 is a succinct description of our method to find a nearly stable model of a normal program P.

---

input: the set $\Pi = \Pi(P)$.
output: a transformation $P' \in ContraCondens(P)$ and a stable model M of P'.
begin

Let $j := 0$
Let $P^j := \emptyset$; let $M^j := \emptyset$; Let $\Pi^j := \Pi$
while Π^j contains a positive clause C containing c_1 derived from
some $C^0 = \{c_1, c_2, \ldots, c_m\} \in \Pi$
$r := c_1 \leftarrow \bar{c}_2, \ldots, \bar{c}_m$;
$P^{j+1} := P^j + r$;
$M^{j+1} := M^j \cup \{c_1, \bar{c}_2, \ldots, \bar{c}_m\}$;
$\Pi^{j+1} := R(\Pi, M^j)$
$j := j + 1$
wend
Let $P'' \in Normal(\Pi - C(P^j))$;
$M' := M^j$;
$P' := P^j \cup P''$
return (P', M');

end;

Figure 1: Finding a suitable transformation of a normal program P

We present an examples to illustrate the application of the method described above.

Example 3.12

Consider the following program:

$P: \; b \leftarrow \neg a$

$$c \leftarrow \neg b$$

$$a \leftarrow \neg c$$

Since $Stable(P) = \emptyset$, we transform P into a set of clauses

$$C(P) = \{\{a, b\}, \{b, c\}, \{c, a\}\}.$$

Now $\Pi = \Pi(P) = C(P)$. Note that Π^0 contains a positive clause $\{c, a\}$; so we let P^1 contain the rule

$$r = c \leftarrow \neg a$$

and

$$M^1 = \{c, \neg a\}$$

Now the clauses $\{a, c\}$ and $\{b, c\}$ can be removed from Π^0 and a can be removed from $\{a, b\}$. Therefore,

$$\Pi^1 = \Sigma^1 = \{b\}$$

Since $\{b\}$ has been derived from $\{a, b\}$, P^2 will include the rule $b \leftarrow \neg a$ $M^2 = \{b, c, \neg a\}$.

Note that now $\Sigma^3 = \emptyset$ and therefore, does not contain a positive clause. Now $\Pi - C(P^2) = \{b, c\}$. So, let $P'' = \{c \leftarrow \neg b\}$. Then

$$\begin{aligned} P' &= P^2 \cup \{c \leftarrow \neg b\} \\ &= \{c \leftarrow \neg a, b \leftarrow \neg a, c \leftarrow \neg b\} \end{aligned}$$

and $M' = \{b, c, \neg a\}$ is a stable model of P'.

¿From Lemma 3.11, by application of the expansion lemma, it follows immediately that

Theorem 3.13 *For every logic program P, $Stable(P + ContraCondens(P)) \neq \emptyset$*

Since $P \models ContraCondens(P)$, applying the Expansion Lemma, we obtain a much stronger result for non-atomic information, stating that every reasoner can use rule information from weaker reasoners:

Theorem 3.14 *For every program P and every semantics $Sem > Sem'$, $Sem(P + \{r \in Rules(P) \mid Sem'(P) \models r\}) \neq \emptyset$*

4 Exploring the Framework

We have only explored a part of the new framework. We will briefly address some more observations:

Disjunctive Programs

We have concentrated on normal programs + constraints. It is not difficult to show, however, that (see [19]) *ContraCondens* additions to general, disjunctive programs are sufficient to provide every (consistent) program with an acceptable model. In that sense, *ContraCondens* operations are more powerful than the shift-operations discussed in [14] to provide disjunctive programs with a (weakly)-stable model.

Skeptical versus credulous revision

The framework offers possibilities to study both skeptical and credulous forms of revision. In this paper we have discussed the skeptical variant by requiring that information should be inferable in every intended model belonging to some class. It should also be possible to use the credulous form of revision by using one or a few intended models belonging to a weaker semantics. In fact is is not difficult to prove that for every credulous reasoner, atomic information from weaker reasoners suffices.

Reducibility by communication

In [19] we have shown that using contrapositives and condensations, it can be shown that stable agents reduce to minimal reasoners, i.e. if a stable reasoner consults his minimal neighbours, the agent cannot obtain more information than by minimal reasoning can be obtained. It is tempting to believe that in our framework this is the general picture: by using information from weaker reasoners a stronger agent is reduced to them.
This, however, is not true if we compare the positivistic semantics and the supported semantics: positivistic agents do not reduce to supported agents if they add contrapositives to the program. Take the following example:

$$P: \quad a \quad \leftarrow \quad \neg a, \neg b$$

$$b \leftarrow c$$
$$c \leftarrow b$$
$$d \leftarrow e$$
$$e \leftarrow d$$

This program does not have a positivistic model. It's supported models are $M_1 = \{\neg a, b, c, d, e\}$ and $M_2 = \{\neg a, b, c, \neg d, \neg e\}$. Therefore, $Supp(P) \models \neg a, b, c$, but $Supp(P) \not\models \neg d, \neg e$. Clearly, $Pos(P+Contra(P)) = \{\{a, \neg b, \neg c, \neg d, \neg e\}, \{\neg a, b, c, \neg d, \neg e\}\}$. Hence, $Pos(P + Contra(P)) \models \neg d, \neg e$.

Direct versus indirect communication

In this paper we have allowed stable reasoners to communicate with classical reasoners directly. It is, however, easy to show that if agents are allowed to consult arbitrarily weaker agents, they weaken their semantics more than if they are forced to consult their nearest neighbours:
Take the previous program P. If the stable agent is allowed to communicate with classical agents, no atomic information can be used since $Mod(P)$ has no atomic consequences. We can add the rules $a \leftarrow \neg b$ and $b \leftarrow \neg a$ to P, since these occur in $Contra(P)$ and thus are classical consequences of P. Then $Stable(P + \{b \leftarrow \neg a,\ a \leftarrow \neg b\}) = \{\{a, \neg b, \neg c, \neg d, \neg e\}, \{\neg a, b, c, \neg d, \neg e\}\}$. But now $Stable'(P) \not\models \neg a$ and can be seen to be weaker than the stable model semantics based on information from supported agents only.

5 Conclusion

We have developed a new framework for doing revision of logic programs, stressing the relations between different non-monotonic semantics and the kind of information needed to find a successful revision of the program.
Our main results were:

1. pointing out an important difference between stable and positivistic (supported) reasoners in terms of the information they need to obtain from weaker reasoners in order to revise their programs.

2. a characterization of the kind of information needed to do successful revision for every reasoner.

3. observing that agent A that consults a weaker agent B does not always derive exactly the conclusions of B; moreover, if $C < B < A$, there may be subtle differences between A consulting C directly, and A consulting B (who then may consult C).

References

[1] N. Bidoit, Negation in rule-based database languages: a survey, *Theoretical Computer Science*, **78**, (1991), 3–83.

[2] J. Dix, G. Gottlob, V. Marek, Causal Models of Disjunctive Logic Programs, in: *Proceedings of the Tenth International Conference on Logic Programming ICLP'94*, 1994.

[3] Doyle, J., A Truth Maintenance System, *Artificial Intelligence* 12, 1979

[4] P. Gärdenfors, *Knowledge in Flux*, MIT Press, Cambridge, MA, 1988

[5] M. Gelfond and V. Lifschitz, The Stable Model Semantics for Logic Programming. In: *Fifth International Conference Symposium on Logic Programming*, pp. 1070-1080, 1988.

[6] L. Giordano and A. Martelli, Generalized Stable Models, Truth Maintenance and Conflict Resolution, in: D. Warren and P. Szeredi (eds) *Proceedings of the 7th International Conference on Logic Programming*, pp. 427-441, 1990.

[7] G. Gottlob, C. G. Fermüller, Removing Redundancy from a clause, *Artificial Intelligence*, **61** (1993), 263–289.

[8] J. W. Lloyd, *Foundations of Logic Programming*, Springer Verlag, Heidelberg, 1987.

[9] W. Marek, V.S. Subrahmanian, The relationship between stable, supported, default and auto-epistemic semantics for general logic programs, *Theoretical Computer Science* 103 (1992) 365–386.

[10] V. Marek and M. Truszczyński, *Nonmonotonic Logic*, Springer Verlag, Heidelberg, 1993.

[11] L. M. Pereira, J. J. Alferes and J. N. Aparicio, Contradiction Removal within well-founded semantics. In: A. Nerode, W. Marek and V. S. Subrahmanian, (eds.), *First International Workshop on Logic Programming and Non-monotonic Reasoning*, MIT Press, 1991

[12] L. M. Pereira, J. J. Alferes and J. N. Aparicio, The Extended Stable Models of Contradiction Removal Semantics. In: P. Barahona, L.M. Pereira and A. Porto, (eds.), *Proceedings -EPIA 91*, Springer Verlag, Heidelberg, 1991.

[13] H. Rott, Modellings for Belief Change: Base Contraction, Multiple Contraction, and Epistemic Entrenchment, in: D. Pearce and G. Wagner, *Logics in AI*, Springer Verlag Berlin, 1992.

[14] M. Schaerf, Negation and Minimality in Disjunctive Databases. In: C. Beeri (ed.), *Proceedings of the Twelfth Conference on Principles of Database Systems (PODS-93)*, pp. 147–157, ACM-Press, 1993.

[15] C. Witteveen and G. Brewka, Skeptical Reason Maintenance and Belief Revision, *Artificial Intelligence*, **61** (1993) 1–36.

[16] C. Witteveen and W. van der Hoek. Belief revision by expansion. In M. Clarke, R. Kruse, and S. Moral, editors, *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 380–387. Springer-Verlag, 1993.

[17] C. Witteveen, W. van der Hoek and H. de Nivelle. Revision of non-monotonic theories: Some postulates and an apllication to logic programming. In D. Pearce C. MacNish and L.M. Pereira, editors, *Logics in Artificial Intelligence, LNAI 838*, pages 137–151. Springer-Verlag, 1994.

[18] C. M. Jonker and C. Witteveen. Revision by expansion. In G. Lakemeyer and B. Nebel, editors, *Foundations of Knowledge Representation and Reasoning*, pages 333–354. Springer Verlag, Lecture Notes in AI Series, Volume 810, 1994.

[19] C. Witteveen. Shifting and Condensing Logic programs. TWI-report 1994, Delft University of Technology, to appear.