Revision by Expansion in Logic Programs

Cees Witteveen[1] and Catholijn Jonker[2]

[1] Delft University of Technology,
Dept. of Mathematics and Computer Science,
P.O. Box 356, 2600 AJ Delft, The Netherlands,
e-mail: witt@cs.tudelft.nl
[2] Utrecht University,
Depts of Philosophy and Computer Science,
Heidelberglaan 8,
3584 CS Utrecht, The Netherlands,
e-mail: cmjonker@phil.ruu.nl

Abstract. We discuss the general problem of revising a contradictory non-monotonic theory and we show that sometimes expanding the theory is more appropriate than contracting it in order to remove the contradiction. We apply this idea of *theory-expansion* to logic programs with negation and constraints.
Using the well-founded (wf-) model semantics for logic programs as our starting point we show that this model may be contradictory due to a clash between the assumption made in the wf-model to consider unfounded atoms to be false and the repercussions constraints can have on this assumption.
Then we show that the contradiction can be removed by adding rules to unfounded atoms in the program. We propose to use the noncontradictory wf-model of such an *expansion* as the semantics of the original program.

We develop a formal framework for program expansion, studying properties as completeness, minimality and computational complexity of expansions.
We think that program expansion is the best framework to study procedurally defined revision processes as proposed in truth maintenance and logic programming such as dependency-directed backtracking and the recently proposed contradiction removal semantics.
Using the framework of program expansions we are able to determine the complexity profiles of these approaches as well as significant generalizations of both of them.

1 Introduction

1.1 Revision of non-monotonic theories

In this paper we deal with revision in logic programs. Such a revision has to occur if the current program becomes contradictory, i.e. has no acceptable model. To remove such a contradiction, we propose to *expand* the current program with new rules in such a way that the expanded program is contradiction-free.

At first sight, expanding a theory in order to remove a contradiction might seem an odd idea. For, in the currently dominant Alchourrón-Gärdenfors-Makinson (AGM-) approach ([1, 4]) to theory revision, instead of expanding a contradictory theory we would *contract* it in order to find a smaller but consistent theory.
The difference, however, between theories dealt with in the AGM-approach and our logic programs is that in the former an underlying *monotonic logic* is assumed, while we have to deal with logic programs in which negation by default can occur, giving it the characteristics of a *non-monotonic theory*.
We will argue that non-monotonicity requires an adaptation of the standard AGM-approach to theory revision.
First of all, although we think that, like revision in standard monotonic theories, revision in non-monotonic theories should be based on consistency as a meta-constraint, we claim that the notion of (in)consistency used has to be generalized. While classically a theory is inconsistent iff it admits no models at all, for an inconsistent non-monotonic theory T it is perfectly possible to have classical models. The reason is that in a nonmonotonic theory not every classical model of the theory is considered *acceptable*. So usually we distinguish for a given theory[3] T a subset of the set $Mod(T)$ of its classical models: its acceptable models.
Consequentially, T is called *(non-monotonically) inconsistent* iff it has no acceptable models (allowing it still to have some classical models). So, classical inconsistency is a kind of limiting case for inconsistent non-monotonic theories.

This notion of non-monotonic inconsistency has some important consequences. Firstly, it disposes of the main rationale for applying revision by contraction if a theory is inconsistent. Whereas retraction in monotonic logic seems perfectly reasonable, if a non-monotonic theory is inconsistent it is not so obvious why we should apply it. For example, if the theory has classical models, we could adapt our notion of acceptability, and turn a classical model into an acceptable one. However, instead of changing the semantics, there is another way to solve a revision problem.
Note that our definition of non-monotonic inconsistency implies that:

> *for an* inconsistent *non-monotonic theory T there may exist* consistent *theories T' containing T.*

[3] Here, a *a theory* T is a set of sentences over a language $\mathcal{L}$. We use $Cn(T)$ to denote the deductive closure of T. In terms of the AGM-theory, T is a belief base and $Cn(T)$ the set of beliefs defined by it.

This suggests that

> *suppose we have a consistent non-monotonic theory T such that $T \cup \{\phi\}$ does not have an acceptable model. Then, instead of contracting T, we could try to find a theory T' containing T such that $T' \cup \{\phi\}$ has an acceptable model.*

Let us call such a theory T' an *expansion* of T.

Therefore, theory-expansion at least is an option in revising contradictory non-monotonic theories. To show that sometimes it is a better alternative, we will discuss a proposal for applying theory-expansion in the revision of *logic programs*.
We will argue that whenever a logic program is (purely) non-monotonically inconsistent, it is more appropriate to apply *theory expansion* than *theory contraction* to solve a revision problem. The reason for this is provided by the special nature of *grounded* reasoning in the semantics of logic programming, governed by the following meta-rule:

> *unless there is a* grounded reason *for a statement, assume it to be false.*

Then, applying revision by expansion is justified by the following consideration:

> *if in a theory T a contradiction arises as a consequence of making some assumptions, it seems more adequate to revise the assumptions than the sentences of T.*

Observe that revising these assumptions means that we have to state *explicitly* that some beliefs cannot be false. As we will see, the only way to express such statements in a logic program is to *add* suitable arguments for such assumptions.

1.2 Plan of the paper

Using logic programs with constraints, first we will give some motivation for introducing expansions as revisions of contradictory programs. Then we will introduce the notion of an expansion function and we will discuss the notion of *completeness* of such functions.
Intuitively, an expansion function E is complete wrt. a class $\mathcal{P}$ of programs if E returns an expansion for a program $P \in \mathcal{P}$ whenever there is an expansion possible for P. So complete expansions are the most successful revisions for a class of programs.
Since it makes sense to concentrate on complete functions only, we would like to represent such functions by expansion functions having some very simple form. We will show that there exists a regular class of expansion functions having this feature (u-t-simple expansion functions) and that these functions can be used to represent and simulate the effect of every possible expansion function.
We will also give computational complexity results for expansion functions. In particular, we will deal with the problem of finding *minimal expansions*, showing that the problem of minimal size and minimal change expansions both are

intractable. If we relax minimality to inclusion minimality it turns out that inclusion minimal size expansions can be found very efficiently, but relaxing the minimal change problem does not change the difficulty of the original problem.

A general problem, however, with these minimal expansions is that they don't need to be unique for a program.
Therefore, in the subsequent sections we discuss a canonical approach to expansions which combines a set of expansions to obtain a uniquely defined non-contradictory wf-model. The wf-model of such a "summarizing" expansion can be proposed as a canonical (non-contradictory) extended stable model of the program.

We show that some canonical expansions can be found effectively, but that in general those canonical approaches based on some form of size or change minimality are intractable, even if they are based on inclusion minimal size expansions.

1.3 Related research

In truth maintenance -a closely related formalism- program expansion has been applied in the context of *dependency-directed backtracking methods* to perform *belief revision* in case a contradiction has been detected ([3, 13, 17]). As these methods mainly have been stated informally and in an procedural way, there are little or no formal results.
In auto-epistemic logic, Morris ([9]) has suggested something like program expansion for auto-epistemic theories that have no extension. The simple idea is: if there is no extension for a set of premises S, then a set-inclusion minimal set of ordinary (i.e. modal-operator-free) premises is added to S such that an extension exists.
In logic programming, the work of Pereira et al. ([10]) on Contradiction Removal Semantics can be seen as a special expansion method, allowing for revision of assumptions.
Also the *abduction problem* in logic programming is closely related to the expansion problem: an abduction problem can be reformulated as the search for a suitable expansion of a program if some constraints are added restricting the admissible truth-value of some atoms.

In none of the references cited above, however, a general characterization of program expansion has been given and also the aspects of completeness and tractability of expansion methods have not been discussed as we will do in this paper.

2 Preliminaries

2.1 Programs and interpretations.

By a finite propositional logic program P we mean a finite set of rules r of the form

$$A \leftarrow L_1, L_2, \ldots, L_m \qquad m \geq 0$$

where A is a propositional atom and $L_1, \ldots, L_m$ are positive or negative (propositional) literals. In a negative literal $\sim B$, the negation operator $\sim$ stands for negation by default.
We will often abbreviate such a rule r by

$$A \leftarrow \Lambda \quad \text{or} \quad A \leftarrow \Lambda^+, \Lambda^-$$

were Λ^+, $((\Lambda^+(r))$ denotes the conjunction of the positive literals, Λ^-, $(\Lambda^-(r))$ the conjunction of the negative literals in the body of r and Λ, $(\Lambda(r))$ is the conjunction of Λ^- and Λ^+.
We will denote the head A of a rule r by $hd(r)$ and $hd(P) = \{hd(r) \mid r \in P\}$. For convenience, we will also use Λ to denote the *set* of literals occurring in the body Λ of a rule.
$\mathcal{B}_P$ denotes the Herbrand Base of P. For a set S of literals $\sim S = \{\sim L \mid L \in S\}$ is also a set of literals, where $\sim\sim L = L$. Let $Lit(P) = \mathcal{B}_P \cup \sim \mathcal{B}_P$.
Let ϕ be a formula over $\mathcal{B}_P$. We use $Lit(\phi)$ $(Lit^+(\phi), Lit^-(\phi))$ to denote the set of all literals(positive literals, negative literals) occurring in ϕ. A three-valued semantics for logic programs will be used distinguishing the truth-values $\mathbf{t}$, $\mathbf{f}$ and $\mathbf{u}$. We use two orderings of these truth-values: the lattice $\mathbf{3}_t$, defined by $\mathbf{f} \leq_t \mathbf{u} \leq_t \mathbf{t}$ and the semi-lattice $\mathbf{3}_k$ defined by $\mathbf{u} \leq_k \mathbf{f}$, $\mathbf{u} \leq_k \mathbf{t}$.
A three-valued interpretation I is a truth-assignment $\mathcal{B}_P \longmapsto \mathbf{3}_t$. We will also use I^x to stand for $I^{-1}(x)$, $x \in \mathbf{3}_t$ and we will also represent I by the (consistent) subset $I^t \cup \sim I^f \subseteq Lit(P)$.
Interpretations are extended to formulas over $\wedge, \sim$ by using the standard strong Kleene interpretations:

$$\wedge : \; I(\alpha \wedge \beta) = min_t\{I(\alpha), I(\beta)\}$$
$$\sim: \;\; \sim\mathbf{t} = \mathbf{f}, \sim\mathbf{u} = \mathbf{u}, \sim\mathbf{f} = \mathbf{t}.$$

The implication operator "$\leftarrow$" is interpreted as *weak* implication, defined by

$$I(A \leftarrow \Lambda) = \begin{cases} \mathbf{t} \text{ if } I(A) \geq_t I(\Lambda) \\ \mathbf{f} \text{ otherwise} \end{cases}$$

The ordering $\leq_k$ between literals can be extended to a knowledge ordering $\sqsubseteq_k$ between interpretations in the usual way:

$$I \sqsubseteq_k I' \;\text{ iff }\; I(A) \leq_k I'(A) \quad \text{for every } A \in \mathcal{B}_P$$

The truth ordering $\sqsubseteq_t$ for interpretations is defined analogously.
Finally, an interpretation I of P is a *model* of P iff $I(r) = \mathbf{t}$ for every rule $r \in P$.

Sometimes we will use the constants $\mathbf{t}$ or $\mathbf{u}$ in the body of program rules. We will assume that the meaning of these constants is respected in the interpretations.

2.2 Well-founded semantics

We will take the well-known *Well-Founded model* $WF(P)$ as the semantics of logic program P[4].
In the sequel we will use the notion of a *well-founded proof* (wf-proof) for an atom A. Such a wf-proof is a set of rules which can be used as an argument for considering A not to be false.

Property 2.1 *For every atom A such that $WF(P)(A) \neq \mathbf{f}$ there exists a smallest finite sequence $\sigma_P(A) = (r_1, r_2, \ldots, r_k)$ of rules in P such that*

1. *$hd(r_k) = A$;*
2. *for every $i \leq k$, $Lit^+(A(r_i)) \subseteq \{hd(r_1), hd(r_2), \ldots, hd(r_{i-1})\}$, i.e. the belief in $hd(r_i)$ is grounded;*
3. *$WF(P)(hd(r_i)) = WF(P)(A(r_i)) \geq_t WF(P)(hd(r_k))$, i.e., the belief in $hd(r_i)$ is supported.*

Such a sequence $\sigma_P(A)$ is called a well-founded proof *(wf-proof) of A in P.*

Such a wf-proof in fact is a simple generalization of the standard notion of a proof:

Lemma 2.2 *Let P be a program. If there is a wf-proof $\sigma_P(A)$ containing only positive rules then $P \models A$, i.e., A is a logical consequence of P.*

2.3 Models of programs with constraints

In normal logic programs using negation as failure it is not possible to express that there exists an incompatibility relation between two or more literals occurring in the program. For example, it is not possible to express that two literals A and B cannot be simultaneaously true.
To express such incompatibility relations, we need constraints. A program with *constraints* is a program P, containing a special subset P_C of rules, called *constraints*. Constraints are represented as rules of the form $\perp \leftarrow A$. The special atom $\perp$ does not occur as antecedent of any rule in P. Such a constraint expresses that the literals occurring in A cannot be true simultaneously.
A program with *constraints* is a program P, containing a special subset P_C of these constraints.

For programs P with a set of constraints P_C, we define a model M of P to be a *consistent* or *C-model* of P iff $M(\perp) \neq \mathbf{t}$. Otherwise, M is *contradictory*.
P is called *C-consistent* if there exists a C-model of P. Otherwise, P is called *C-inconsistent*.
We define M to be the *wf_C-model of P* if $M = WF(P)$ and M is consistent.

[4] We refer to [7, 16, 18, 21] for exact definitions and relations to other possible semantics for logic programming; here we only mention its relation to the three-valued *stable model semantics*: the Well-Founded model $WF(P)$ is the unique $\sqsubseteq_k$-least stable model of P.

3 Revision by Expansion

For every *general* logic program the wf-model $WF(P)$ is uniquely defined. For programs with constraints however, not every program P does have a wf_C model. For example, the program

$$\begin{aligned} P_1 \ : p &\leftarrow \\ q &\leftarrow p, \sim r \\ \bot &\leftarrow q \end{aligned}$$

does not have a wf_C-model, and the same is true for the the program

$$\begin{aligned} P_2 \ : p &\leftarrow \\ \bot &\leftarrow p \end{aligned}$$

We argue that in the first case there is a very natural solution to the problem: although the wf-model $M = \{p, q, \bot, \sim r\}$ for P_1 is contradictory, there are other models of P_1 which are not. For example, $\{p, r, \sim q, \sim \bot\}$ and $\{p\}$ are three-valued C-models of P. The reason why M was selected by the wf-semantics is, of course, that there is no direct grounded reason to consider r as true or unknown. Hence, r is *assumed* to be false and this causes the constraint $\bot \leftarrow q$ to be violated.
But of course, isn't the violation of such a constraint a perfect reason to consider r *not* to be false? Hence, we see that there exists an *indirect* reason to consider r as true or unknown, which cannot be detected by the wf-semantics. Now in order to express that we want to assign some truth to an atom A in a logic program, we have to add a rule for it.
So it seems that we can repair such a defect of the wf-semantics by recognizing these *repercussions* of the wf-requirements in the presence of constraints by

(i) adding some reason (rule) for one or more such assumptions r;
(ii) determining the wf-model of the obtained *expansion* of the original program.

Hence, we see that in this case, in which P has a non-empty set of C-models, a theory-expansion approach can be motivated in a simple way.

On the other hand, in case of P_2, the program itself is C-inconsistent and the wf-model semantics cannot be blamed for a defect. Here, in fact, every expansion of P_2 is C-inconsistent, so there exists no expansion having a wf_C-model.
In this latter case, standard theory revision methods could be applied, including the application of *contraction* operations (cf. e.g. [4]). However, as mentioned in the introduction, we will restrict our attention to expansion methods.

Notation: A program P expanded by a set of rules R is denoted by $P + R$. If R is a singleton $R = \{r\}$, we will also use $P + r$.

In [21] we have investigated some relations between the well-founded model of a program P and the well-founded model of certain expansions of P.

Here we will need only one property, needed in the subsequent sections.
This property implies not only *that* rules of the form $A \leftarrow \mathbf{u}$ can be added to the program in order to find a $\sqsubseteq_k$-smaller well-founded model of the expansion, but also *when* such an addition will result in a weakening of the wf-model.

Property 3.1 (u-anti-monotonicity) *For every P and $A \in \mathcal{B}_P$:*
(1) $WF(P + \{A \leftarrow \mathbf{u}\}) \sqsubseteq_k WF(P)$
and,
(2) if $WF(P)(A) \geq_t \mathbf{u}$ then $WF(P) = WF(P + \{A \leftarrow \mathbf{u}\})$.

4 Expansions and Expansion functions

We will now discuss a framework for program expansion.

Definition 4.1 *An* **expansion** *of a program P is a program P' such that $P \subseteq P'$ and $P_C = P'_C$.*

Definition 4.2 *A* **wf-expansion** *of a program P is an expansion P' of P such that P' has a wf_C-model.*

Our interest is also in computational aspects of program expansion. Therefore we want to study the complexity of algorithms which given a program would find a suitable expansion of it.
So we will study expansions of programs by studying properties of expansion *functions* that are used to generate them.

Definition 4.3 *Given a class of programs $\mathcal{P}$, a* **wf-expansion function** *is a (partial) computable mapping E, whose domain is a class of programs, assigning to every $P \in dom(E)$ a wf-expansion $E(P) = P'$ of P.*
If $P \in dom(E)$, $M_E(P)$ denotes the **wf-expansion model** *$WF(P')$ returned by E.*

Example 4.4 Consider again the program

$$\begin{aligned} P: p &\leftarrow \\ q &\leftarrow p, \sim r \\ \bot &\leftarrow q \end{aligned}$$

Suppose we have an expansion function E, which applied to a program P, expands P by adding a rule $s \leftarrow$ for every negative literal $\sim s$ occurring in P.
Then E will return the expansion $P' = P + \{r \leftarrow\}$ and finds the wf_C model $WF(P') = \{p, r, \sim q, \sim \bot\}$ as the wf-expansion model for P.
If, however, the rule $q \leftarrow r$ is added to P, E is not defined for $P \cup \{q \leftarrow r\}$: adding $r \leftarrow$ to this program now results in a program P' having no wf_C-model, since $WF(P') = \{p, r, q, \bot\}$, so $WF(P')(\bot) = \mathbf{t}$.
If we take another function E' that always adds a rule $s \leftarrow \sim s$ for every negative literal $\sim s$ occurring in the body of some rule, E' is also defined for this latter program, since then $WF(P') = \{p\}$. ■

4.1 Revisability and Completeness

As the example given above shows, some expansion functions may succeed for some programs but not for others. Clearly, given an arbitrary program P we would like to know whether there exists *some* function for a successful expansion of *this* P. This motivates the following definition:

Definition 4.5 *A program P is called* **revisable** *iff there exists a wf-expansion function E, such that $P \in dom(E)$.*

The following simple result shows that a program P is revisable iff, classically, it is non-contradictory:

Theorem 4.6 *A program P is revisable iff P has a C-model.*

As a consequence, another characterization of revisable programs can be given by means of wf-proofs of $\perp$:

Corollary 4.7 *A program P is revisable iff every wf-proof $\sigma_P(\perp)$ contains at least one rule r having at least one negative antecedent.*

These results imply that the domain of a *suitable* expansion function E should contain every C-consistent program. Functions satisfying this requirement will be called *complete*.
More exactly, let $\mathcal{P}$ be a class of programs and let

$$CONS(\mathcal{P}) = \{P \in \mathcal{P} \mid \text{P is a } C\text{-consistent program }\}$$

Then we define completeness as:

Definition 4.8 *A program expansion function E is said to be* **complete** *with respect to $\mathcal{P}$ iff $dom(E) = CONS(\mathcal{P})$.*

Clearly, it makes sense to concentrate on complete functions, since they can be considered as the most successful expansion functions.

4.2 Complete classes of expansion functions

We will show now that every (complete) expansion function can be represented by some subset of atoms from $\mathcal{B}_P$. First, we will prove that every expansion function can be simulated by a some *simple* expansion function. Then we will show how to reduce simple expansion functions to subsets of $\mathcal{B}_P$.

Suppose we could represent the effect of every conceivable expansion function by using functions in a rather small and regular class of expansions. This latter class then should have the property that the effect of every conceivable expansion function on atoms in $\mathcal{B}_P$ can be *simulated* by some member of this class. If such a class $\mathcal{E}$ has this property, we will call it a *complete class*:

Definition 4.9 *Let $\mathcal{E}$ be a class of expansion functions and $\mathcal{P}$ a class of programs. $\mathcal{E}$ is said to be a* **complete class of expansion functions with respect to** $\mathcal{P}$ *iff for every expansion function E' there exists an expansion function $E \in \mathcal{E}$ such that*

1. $dom_{\mathcal{P}}(E) = dom_{\mathcal{P}}(E')$
2. *for every* $P \in dom_{\mathcal{P}}(E')$, $WF(E(P)) =_{Lit(P)} WF(E'(P))$.

Here, $dom_{\mathcal{P}}(E)$ refers to the domain of E restricted to the elements from $\mathcal{P}$ and $=_{Lit(P)}$ refers to equality restricted to elements occurring in $Lit(P)$.
We will now define a complete class of regular expansion functions.

Definition 4.10 *An expansion function E is called* **simple** *if for every $P \in dom(E)$ only rules of the form $A \leftarrow \mathbf{t}$ or $A \leftarrow \mathbf{u}$ are added to P, where $A \in \mathcal{B}_P$. We will call E* **t-simple** *if only rules of the form $A \leftarrow \mathbf{t}$ are added and* **u-simple** *if only rules of the form $A \leftarrow \mathbf{u}$ are added.*

The following lemma shows that the class of simple expansion functions is a class of complete expansion functions with respect to the class of all propositional logic programs with constraints.

Lemma 4.11 *Let E be an* arbitrary expansion function *and P an arbitrary element of $dom(E)$. Then there exists a* simple expansion function E_{simple} *such that*

1. $dom(E_{simple}) = dom(E)$ *and*
2. $WF(E_{simple}(P)) =_{Lit(P)} WF(E(P))$.

Note that given a program P, the effect of a **t**- or **u**-simple expansion function can be represented by set of conclusions of the rules added to P. In general, given an expansion function E and a program P, we will call the set $E_P = \{hd(r) \mid r \in E(P) - P\}$ an *expansion set* of P.

The following observation implies that we don't lose completeness in representing complete simple expansion functions by **u**-simple expansion sets:

Observation 4.12 *For every complete simple expansion function E, there exists a complete* **u***-simple expansion function E' such that for every $P \in dom(E)$, $E_P = E'_P$.*

Taking these results together, we have the following theorem:

Theorem 4.13 *For every complete expansion function E' there exists a* **u***-simple expansion function E, such that $dom(E') = dom(E)$.*
In particular, the **u***-simple function E such that for every P, $E_P = E'_P \cap \mathcal{B}_P$ suffices.*

This result, and the completeness of simple expansion functions, show that we can study properties of arbitrary complete expansion functions E by reducing them to the expansion set $E_P \cap \mathcal{B}_P$.
In the subsequent sections we will assume that every expansion function E has been reduced to such a set and we will simply use E_P to denote this expansion set, further reducing it to E if P is understood.

4.3 Complexity of complete expansion functions

At this point the reader might ask how difficult it might be to actually compute complete expansion functions.
It turns out that finding an arbitrary expansion of a revisable program is tractable. In fact, computing $WF(P)$ and selecting the atoms evaluated false by $WF(P)$ already suffices.

Theorem 4.14 (tractability of arbitrary expansions)
There exists an $O(|P|^2)$-time computable expansion function for programs with constraints.

Proof Sketch. Let E be the expansion function returning for every revisable but contradictory program P with constraints the expansion set

$$E_P = \{A \mid WF(A) = \mathbf{f}, A \in \mathcal{B}_P\}.$$

It is easy to see that $P+E_P$ is contradiction-free. Since revisability can be checked in $O(|P|)$-time and the WF-model of $P+E_P$ can be computed in $O(|P+E_P|^2) = O(|P|^2)$ time, the theorem follows.

Remark. At this point, the reader might ask, why we did not require an expansion model to *falsify* $\perp$, i.e., to require that $M(\perp) = \mathbf{f}$ for every wf_C model of P. Such a C-model could be called satisfying *hard* constraints.
Among others, the reason is that allowing hard constraints would make the theory less interesting from a computational point of view: we can show that hard constraints already render the problem to find an arbritrary wf-expansion of P intractable.

Proposition 4.15 *Let P be a program with at least one hard constraint. Then finding an arbitrary wf-expansion of P is NP-Hard.*

5 Minimal Expansion Methods

We only expand a program if it is necessary to do so. And if it is necessary it seems natural to try to change the program as little as possible.
This idea of minimality can be specified either in a *syntactical* or in a *semantical* way.

1. *Syntactically* an expansion function is a minimal expansion if it minimizes the amount of expansion rules added to a program P i.e., the *size* of the expansion, in order to obtain a contradiction-free expanded program.

   Therefore, we will call an expansion function E a *minimal size expansion* function if for every E' and $P \in dom(E) \cap dom(E')$, we have

   $$|E(P)| \leq |E'(P)|$$

2. *Semantically*, an expansion function E is a minimal expansion if it affects the information conveyed by the original program in a minimal way. Such a minimal *change* expansion we define as a minimization of the model-difference between the (inconsistent) wf-model of P and the (consistent) wf-model of the expansion $E(P)$.
   More exactly, if $E(P)\Delta P$ is defined as
   $$E(P)\Delta P = \{a \in \mathcal{B}_P \mid WF(E(P))(a) \neq WF(P)(a)\}$$
   then E is a *minimal change* expansion function, if for every E' and $P \in dom(E) \cap dom(E')$, we have
   $$|E(P)\Delta P| \leq |E'(P)\Delta P|$$

As we already remarked we aim at complete and hopefully tractable expansion functions.
It is, however, very unlikely that there exist complete and minimal expansion functions which also can be computed efficiently. If we look at their associated search problems (to find a size- or change-minimal expansion for a given program), these problems turn out to be NP-Hard.

Theorem 5.1 *The problem to find a minimal size or minimal change expansion for an arbitrary program P with constraints is NP-Hard.*

The proof of these results[5] is given by a polynomial turing-reduction from the NP-complete Hitting Set problem (see [5]). Let (S, C, K) be an instance of the Hitting Set problem, where S is a finite set, C is a set of non-empty subsets of S and K is a positive integer.

We can show that this instance has a hitting set of size $\leq K$ iff the program

$$P_{HS} = \{\bot \leftarrow \sim s_{i_1}, \ldots, \sim s_{i_n} \mid \{s_{i_1}, \ldots s_{i_n}\} \in C\}$$

has a minimal expansion of size $\leq K$ or a minimal expansion making at most K changes.
To measure the degree of NP-Hardness more precisely, we can use the technique of *bounded query evaluation* [19], where the complexity of a problem is measured in terms of the necessary number of calls to an NP-oracle in order to solve the problem by an otherwise polynomial algorithm.
The associated decision problems of these problems: *do there exist minimal expansions of size/change K or less?* both are NP-complete problems. By making $O(\log n)$ calls to an NP-oracle for these problems, we can decide the *size* K of the minimal size expansion or the *number* K of changes required by a minimal change expansion, where n is the number of atoms occurring in P.
This number K can be used to determine which atoms will be required to occur in a minimal size- or change-expansion by making $O(n)$ calls to an NP-oracle

[5] For the complete proof, the reader is referred to a technical report [21]

for the associated decision problems. Hence, both problems can be solved by making *at most* $O(\log n) + O(n) = O(n)$ calls to an NP-oracle.
On the other hand, there are more than a logarithmic number of calls necessary, since it is easy to see that if we only allow for $O(\log n)$ calls to an NP-oracle to solve the minimal expansion problem, the existence of any algorithm solving these problems in polynomial time would immediately imply that P = NP[6].
So we have:

Theorem 5.2 *The problem to find a minimal size or minimal change expansion for an arbitrary program P with constraints can be solved by a polynomial algorithm making more than $O(\log n)$ but no more than $O(n)$ calls to an NP-oracle.*

Note that until now we have only analysed *cardinality* minimal size/change expansion functions. It is tempting to relax these notions and to investigate the complexity of their relaxation looking at *inclusion* minimal (size/change) expansion functions.

We will investigate the complexity of these problems separately.

5.1 Inclusion minimal size expansion is easy

For a relaxation of size minimal expansions we do not look at the *number* of elements occurring in a size-minimal expansion, but at the subsets of the expansion verifying that none of them makes the program contradiction-free.

Definition 5.3 *An expansion set E is called an* inclusion size minimal expansion *for a program P if $P + E$ is contradiction-free and there is no strict subset E' of E such that $P + E'$ is contradiction-free.*

It turns out that relaxation of size minimality is a good idea.

Theorem 5.4 *The problem to find an inclusion minimal minimal expansion for an arbitrary program P with constraints can be solved in polynomial time*

Proof Sketch. Without loss of generality assume P to be revisable. Take an arbirary expansion set E for P; We know (see Theorem 4.14) that such an expansion set can be found in polynomial time.
Basically, the idea is to remove elements from E as long as the resulting program $P + E$ still has a WF-model. The following algorithm creates such a minimal expansion E_{min} from a given expansion E:

```
MinExpansion($P$, $E$):
begin
    $E_{min} := E$ ; $i := 1$ ;
    while $i \leq |E|$
```

[6] For an application to a related problem see [6]

```
        if $WF(P + (E_{min} - \{a_i\}))$ is non-contradictory
            then $E_{min} := E_{min} - \{a_i\}$
        fi;
        $i := i + 1$;
    wend;
    return $E_{min}$;
end;
```

The correctness of this algorithm can be derived from the u-antimonotonicity of the WF-operator (see Property 3.1). Since the wf-model of a propositional program can be computed in $O(n \times |P|)$-time, where n is the number of atoms occurring in P (see [22]), *MinExpansion*(P, E) can be computed in $O(n \times (n \times (|P| + n))) = O(n^2 \times |P|)$ time.

Example 5.5 Consider the following program:

$$\begin{aligned} P\ :\ & e \leftarrow \sim a \\ & e \leftarrow \sim a, \sim b \\ & d \leftarrow e \\ & e \leftarrow d, \sim c \\ & \bot \leftarrow e. \end{aligned}$$

The wf-model of this program is $M = \{\sim a, \sim b, \sim c, e, d, \bot\}$.
Take $E = \{a, b, c\}$. Computing *MinExpansion*(P, E) we find $E_{min} = \{a\}$. Hence, $P \cup \{a \leftarrow \sim a\}$ is a minimal size expansion of P.

5.2 Inclusion minimal change expansion is hard

To define a relaxation of minimal change expansion requires some more work. First of all, if we would define E to be an inclusion minimal change expansion if no subset of E is a minimal change expansion, we would immediately identify it with inclusion minimal size expansions.
Note, however, that if E is an expansion set, the set $(P + E)\Delta P$ is also an expansion set. Moreover, it is easy to see that for every expansion set E' contained in $(P + E)\Delta P$, we have:

$$(P + E')\Delta P \subseteq (P + E)\Delta P$$

Therefore, we can relax the change minimality of E with respect to the set $(P + E)\Delta P$ and we define inclusion minimal change expansions as follows:

Definition 5.6 *An expansion set E of P is an* inclusion minimal change expansion *if there is no subset E' of $(P+E)\Delta P$ such that* $|(P+E')\Delta P| < |(P+E)\Delta P|$.

Note that again, every cardinality based minimal-change expansion also is an inclusion minimal change expansion.

The disappointing result however, is, that even relaxation does not help in making the problem easier:

Theorem 5.7 *Given an arbirary program P and an expansion E of P the problem to find an inclusion minimal change expansion is NP-hard.*

And even looking at a more detailed level it turns out that such E-minimal change problems can be solved by polynomial algorithms using at most $O(n)$ but more than $O(\log n)$ calls to an NP-oracle.

6 Canonical Expansion Methods

In the previous section we discussed syntactical and semantical minimality criteria for expansion methods. One of the aspects evolving from minimization-activities is that generally several minimal expansion are associated with a program.
As is common in the research of logic programming, a canonical method is desirable. The most obvious way to attain canonicity is to create a most general expansion method, i.e. an expansion method that contains every expansion method of a certain class. Before studying canonicity in the context of minimality we will first characterize canonicity for arbitrary complete expansion classes.
The most general expansion method that exists is the set of all atoms A such that $WF(P)(A) = \mathbf{f}$ as expansion set for a revisable program P. Obviously, this set can be computed in constant time if the wf-model of P is given and in $O(|P|^2)$-time otherwise. However, this method is in most cases too general, atoms may get changed that do not occur in any wf-proof of $\perp$.
So, let us restrict ourselves to methods that only change atoms that are involved, one way or another, with some wf-proof of $\perp$. These expansion methods will be called *relevant* expansion methods. Note that every minimal size expansion method is relevant.
Restricted to relevancy, we can now obtain an easy method for generalization of such a class of relevant complete expansion methods:
Let $\mathcal{R}$ be a class of relevant complete expansion methods, then

$$R_k = \bigcup\{E \mid E \in \mathcal{R}\}$$

is a relevant and complete expansion method with the feature that for all expansions $E \in \mathcal{R}$ and arbitrary revisable programs P,

$$WF(R_k(P)) \sqsubseteq_k WF(E(P)).$$

Therefore, we call these most general relevant complete expansion methods *knowledge-minimal* expansion methods.

6.1 Knowledge-minimal Expansion Methods

In the above definition of knowledge-minimal expansion methods, such an expansion method is defined with respect to a specific, but arbitrary class of relevant complete expansion methods. Therefore, we can study the knowledge-minimal expansion method for every relevant and complete class. Since first computing every expansion for a revisable program within a specific class R will most often lead to unnecessary high time-complexities, it makes more sense to try to characterize the set of atoms affected by the expansions in R in direct way. In recent years several classes of relevant complete expansion methods have been characterized by their knowledge-minimal expansion and some of these can be computed in cubic-time, given the size of the program, since the associated expansion-sets can be found in linear time. We refer to Section 4.3 where we prove that there exist tractable knowledge-minimal expansion methods. Furthermore we refer to [7, 20, 21, 22] where knowledge-minimal expansions of several classes are introduced and compared. In this paper we only want to stress that knowledge-minimality is in many cases tractable. However, in the next section we will study the knowledge-minimality of the classes of minimal expansion methods and their complexity. This way the lower and upper bounds on the complexity of canonical expansions will be fully determined. We hope that by way of future research, this will help to establish exactly where the narrow line between tractability and intractability is.

6.2 Canonical Minimal Expansion Methods

In this section we will study the knowledge-minimal expansions of the different classes of minimal expansion methods. We will abbreviate knowledge-minimal minimal expansion methods by the term canonical minimal expansion methods. Doing this we generalize an idea suggested in [10] to use a canonical minimal expansion function in order to obtain a canonical wf-expansion for a program with constraints.
We will not discuss the somewhat intricate details of the CRS Pereira et al. proposed, but in the Section 7 we will present a characterization framework for arbitrary expansion methods with we will analyse the CRS.

We will start by considering the abstraction level where the kind of minimality is not important, later we will concentrate on inclusion-minimality of size, since it is the only tractable minimality criterion, then we will continue by relating the complexities of the notions minimality of size and (inclusion) minimality of change to the complexity of inclusion-minimality of size.

As mentioned in Section 6.1, given the a specific sort of minimality and its associated class R, the computation of R_k by first computing all expansions in R will not lead to tractable algorithms. Therefore, we will first compute the complexity of finding a canonical minimal expansion given a specific, but arbitrary, expansion function.
Assume that we have some complete expansion function E and a program $P \in dom(E)$. Overloading E we let $E \subseteq \mathcal{B}_P$ represent the expansion set generated by the expansion function and we will call E the *candidate set.* E_u will denote the set of rules $E_u = \{e \leftarrow \mathbf{u} \mid e \in E\}$.
Every subset E' of E that is also an expansion set, will be called a *subexpansion* of E. Among them are subexpansions that are *minimal subexpansions* of E. Since

we have no reason to prefer one above another we take the union of all minimal subexpansions of E as the *canonical expansion set* E^* associated with E, i.e. if, for example, R denotes the class of all inclusion minimal-size expansions that are subexpansions of E, then E^* would coincide with R_k. Subsequently, $WF(P+E^*_u)$ is called the *canonical wf-expansion model* of P wrt. E.
Given the candidate set E, the canonical wf-expansion model is consistent:

Proposition 6.1 *Let E be a candidate set for a program P. Then E^* is an expansion set for P.*

This follows immediately from the fact that E^* can be defined as the set of elements $e \in E$ for which there exist a minimal subexpansion containing e. Hence, $E^* \subseteq E$. Now the result follows as an easy consequence of E being an expansion set and Property 3.1.

Until now, we made no reference to a specific type of minimality, but in Section 6.2 we will first consider inclusion minimality of size.

Complexity of canonical minimal expansions
Since we know that inclusion minimal size expansion problems are tractable, we might expect the *canonical* inclusion minimal size expansion also to be tractable. We can show, however, that finding such a canonical expansion is intractable. This proof is based on the intractability of the following decision problem:

> *Given a program P, an expansion set $E \subseteq \mathcal{B}_P$ and an element $e \in E$, is there a minimal subexpansion E' of E containing e?*[7]

Note, that we can find the canonical subexpansion E^* of E by quering an oracle for this NP-complete problem $O(|E|)$ times: for every $e \in E$, decide whether or not e occurs in a minimal subexpansion of E. Hence,

Theorem 6.2 *Given an arbitrary program P and a complete expansion set E for P, the problem to find a canonical inclusion minimal-size expansion E^* of E can is NP-hard but can be found by quering an NP oracle $O(|E|)$ times.*

From Section 5 we know that all three other criteria for minimality (of size, of change and inclusion minimal change) are essentially more difficult than inclusion minimal size. Therefore, it is easy to understand that the complexities of canonical minimal-size and canonical inclusion minimal change expansion cannot be of a lower complexity than canonical inclusion minimal size expansion.

7 Analysis and Characterization of Expansion Methods

In this article we discussed several aspects by which expansion methods can be classified: completeness, relevance, (inclusion) minimality of size and change,

[7] So finding an arbitrary minimal size expansion is tractable, but looking for a specific minimal size expansion is intractable.

tractability and canonicity. In this section we will give the analysis two proposals from literature for program expansion that have not been subjected to such an analysis before. We chose the original *Dependency-directed backtracking* method of Doyle [3] and the recent *Contradiction Removal Semantics* of Pereira et al. For a similar analysis of other expansion methods we refer again to [7, 20, 21, 22], see also Section 6.1.
We chose dependency-directed backtracking (DDB) since Doyle was the first to perform revision by expansion by way of DDB. We chose the CRS, since the CRS is the latest, as far as we know, of such proposals other than our own.
We will only go into the details of DDB, when it is absolutely necessary for the understanding of our analysis. A lot of research has already been done concerning DDB, see [3, 7, 8, 13, 17], so that we feel justified in summarizing those results relevant for our analysis. The non-obvious details of the analysis of DDB can be found in [22] and [8]. We will discuss the analysis of CRS in more detail since apart from the authors, i.e. Pereira et al., all research concerning CRS can be found in [21].

7.1 Dependency-directed backtracking

Dependency-directed backtracking was developed by Doyle [3] for the restoration of consistency in Truth Maintenance Systems, which were also developed by Doyle. Truth Maintenance Systems were developed to record reasons for belief, called justifications, and a consistent model for a set of such justifications. Since a complete line of reasoning is recorded, consistency restoration (or revision) must affect as little as possible of the justifications so as not invalidate the greater line of reasoning. Given a wf-proof $(r_1, \ldots, r_m)$ for $\perp$, Doyle, therefore, developed his DDB so that the ensuing expansion set contained elements from the rules of the wf-proof that have the highest possible index. Doyle's priority was therefore a combination of size- and change-minimality.
For reasons of tractibility, DDB was a heuristic for this aim, meaning that, possibly wrong, choices were made in creating the program expansion. This means that several DDB-strategies existed. The best that could happen was that the first choice was right, which implies that then the expansion was size-minimal. Unfortunately, change-minimality was not guaranteed if the first choice was right, see [8]. The worst that could happen was that exponentially many wrong choices were made before an expansion was found, the created expansion was then far from minimal in any sense.
The making of choices also implies that DDB was not a canonical method. However, Doyle ensured that only relevant choices are made and that, if the program is revisable, an expansion is found.
Summarizing, DDB is:

1. complete,
2. relevant,
3. a heuristic for a combination of size- and change-minimality,
4. best case: cubic time, and size-minimal,

5. worst case: exponential time,
6. not canonical.

If we now apply the result of Section 6 to find a reconstruction of DDB that corresponds better to our enhanced insight in the declarative semantics of logic programming, then the first obvious step is to change using a particular DDB-strategy into using all possible DDB-strategies at the same time. In this way we do not have to make choices, but we can instead look for the set E_{ddb} of those atoms for which there exists a DDB-strategy that would have chosen these atoms. This ensures canonicity. The great advantage is that we could prove that we do not need the individual DDB-strategies to determine E_{ddb}, but that we can determine E_{ddb} in linear time. The details can be found in [7, 21] and [22]. Summarizing, under the new reconstruction of DDB, E_{ddb} is:

1. complete,
2. relevant,
3. a combination of size- and change-minimality,
4. tractable: taking cubic time,
5. canonical.

7.2 Contradiction Removal Semantics

The Contradiction Removal Semantics is a canonical expansion function that computes the *canonical expansion set* E^*_{crs} associated with a specific expansion set E_{crs}. Pereira et al. propose the wf-model of E^*_{crs} as a canonical (non-contradictory, extended stable) model of the program.

So the first characterization is that CRS consists of two stages: the first is the computation of the candidate set E_{crs} and the second is the construction of its associated canonical expansion set.
Since the wf-model of a program P can be found in $O(n^2)$ time and the authors state that the expanded program can be obtained by a simple transformation from the original one, this might suggest that finding a "CRS"-model of P is tractable.

In Section 6.2 we showed that the complexity of the second stage is NP-hard. This makes the existence of a tractable algorithm for CRS unlikely. However, before we can make a definite estimation of the time-complexity of the entire CRS, we must first understand the first stage of CRS, i.e. the computation of the candidate set E_{crs}.
The CRS of Pereira et al. constructs its set of candidate literals using a set of so-called *support sets* and a set of *assumption sets*, the union of the latter giving the set of candidates E_{crs}. We will give a brief overview.

Let P be a program and $M = WF(P)$ a contradictory wf-model. Informally, a *support set* $SS(L)$ for a literal L contains all the literals that are used to establish the truth of L in M.

Definition 7.1 (Support Sets) *Let P be a GLP, $M = WF(P)$, $L \in B_P \cup {\sim}B_P$. The* set of support sets $SSS(L)$ *is defined inductively as: If $L \in B_P$, then*

$$SSS(L) = \{Lit(A) \cup \bigcup_{L' \in A} SS(L') \mid L \leftarrow A \in P,\ M(A) = \mathbf{t},\ SS(L') \in SSS(L')\}$$

and if $L \in {\sim}B_P$, then

$$SSS(L) = \{\{{\sim}L'\} \cup SS({\sim}L') \mid \begin{array}{l} {\sim}L \leftarrow A \in P,\ L' \in A,\ M(L') = \mathbf{f}, \\ SS({\sim}L') \in SSS({\sim}L')\} \end{array}$$

An element of $SSS(L)$ is called a support set *and denoted by $SS(L)$.*

Let $SSS(X)$ denote the set of support sets of X. Given a support set $S \in SSS(X)$, an *assumption set* $Ass_S(X)$ is defined as:

$$Ass_S(X) = \{A \in B_P \mid {\sim}A \in S \text{ and } A \notin hd(P)\}$$

Let $ASS(\perp)$ denote the set of assumption sets of $\perp$.

In the original formulation of CRS, the set of candidates is $E_{crs} = ASS(\perp)$, then the set

$$R = \begin{cases} \emptyset & \text{if } \emptyset \in ASS(\perp) \\ \{S^* \mid S \in ASS(\perp)\} & \text{otherwise} \end{cases}$$

is computed and then E^*_{crs} is defined as $\bigcup R$.
It is not difficult to see that, indeed, the CRS is a special canonical revision function. Now that the first stage of the CRS is duly described we proceed to investigate its time-complexity.

Support and Assumption Sets are difficult to compute
Although it is not mentioned explicitly, one could have the impression that the support and assumption sets are indispensible for the computation of the canonical expansion E^*_{crs}. Of course then, one could ask how difficult it is to compute them for an arbitrary program P.

Property 7.2 *[21] There are programs P for which $SSS(\perp)$ and $ASS(\perp)$ cannot be computed in polynomial time.*

As a very easy example, take the following program:

Example 7.3 Let P be the following program:

$$P = \{\perp \leftarrow {\sim} q_0\} + \{q_i \leftarrow p_{i+1}, q_{i+1}\ \ ,\ q_i \leftarrow q_{i+1} \mid i = 0, 1, 2, \ldots, n\}$$

The support sets of $\perp$ are of the form $\{\sim q_0\} \cup SS(\sim q_1)$. For $\sim q_i$ the support sets are of the form $\{\sim p_{i+1}, \sim q_{i+1}\} \cup SS(\sim q_{i+1})$ and $\{\sim q_{i+1}\} \cup SS(\sim q_{i+1})$ for all support sets $SS(\sim q_{i+1})$ of q_{i+1}.
Hence, there are $O(2^n)$ support sets for $\perp$ in this program of size $O(n)$. The number of assumption sets equals the number of subsets of $\{p_1, p_2, \ldots, p_{n-1}\}$ which is $O(2^n)$.

So, it is possible that the first stage of CRS takes exponential time. Although we cannot fundamentaly lower the complexity of the first stage, we can make some improvement:
We can also define the set E_{crs} of candidates as follows:

$$E_{crs} = \begin{cases} \emptyset & \text{if } \emptyset \in ASS(\perp) \\ \bigcup ASS(\perp) & \text{otherwise} \end{cases}$$

The above improvement is obviously not enough since we still need to compute all support sets of $\perp$, of which there could be exponentially many. However, it might be argued that this is not the conclusive answer. For, there might exist a polynomial function to compute E^*_{crs} *without* using the set of assumption sets. However, even under this restriction, the existence of such a function - modulo P$\neq$NP - is impossible.

We proved this result by showing that, given a program P, the problem whether or not a given atom belongs to E^*_{crs} is NP-Hard. Clearly, if this decision problem is NP-Hard, the construction of E^*_{crs} must be NP-Hard as well.

Theorem 7.4 *[21] Given an arbitrary program P and an arbitrary atom $s \in B_P$, the problem whether or not s belongs to E^*_{crs} is NP-Hard.*

So, the second characterizing remark we can make about the CRS, is that additional information used in the CRS cannot be used to speed us the canonical revision process, thus leaving the complexity of CRS to be at least NP-hard.
The third aspect we want to investigate is completeness.
We show that revisable programs exist for which the CRS in its present form[8] is unable to find a wf-expansion.

Example 7.5 Consider the program

$$\begin{aligned} P \ : \ & \perp \leftarrow \sim p, \\ & \perp \leftarrow \sim q, \\ & p \leftarrow q, \\ & q \leftarrow p \end{aligned}$$

Note that $SSS(\perp) = \{\{\sim p, \sim q\}\}$ and $ASS(\perp) = \{\emptyset\}$. Hence, the CRS is not able to find a wf expansion, while, for example, the candidate set $E = \{p, q\}$ and the canonical expansion $E^* = \{p, q\}$ suffice.
The fourth aspect is that CRS comes nowhere near minimality of change. This is obvious from the the definitions of support- and assumption sets. However, the authors never meant CRS to be minimal with respect to change.
The fifth aspect is related to the canonicity of the approach. The CRS is obviously not a minimal expansion revision.
The sixth aspect is that by definition the CRS approach tends to be some kind of minimal-knowledge approach. However, since there are several complete

[8] The CRS is currently under revision, according to [12].

minimal-knowledge expansions that are tractable, the relative worth of CRS is difficult to establish.
We summarize that the CRS is:

1. incomplete;
2. relevant;
3. not a minimal-change expansion method;
4. intractable;
5. canonical

In [21] this analysis appears in full and, based on the analysis, the CRS is generalized so that completeness can be ensured.
Based on these analysis, we conclude that DDB is based on good principles and lends itself to tractable computations, while the position of CRS is not clear with respect to the minimality criteria and cannot lead to a tractable algorithm.
As far as minimality of change is concerned, we can compare DDB and CRS by introducing:
Let E be an expansion set and let $\hat{E} = P + E \Delta P$. Some simple facts are:

- if E is minimal then $E \subseteq \hat{E}$;
- for every B, if $B \subseteq \hat{E}$ then $\hat{B} \subseteq \hat{E}$.

In [21] we proved that $E_{ddb} \subseteq \hat{E}_{crs}$ which implies that $\hat{E}_{ddb} \subseteq \hat{E}_{crs}$ and thus $\hat{E}^*_{ddb} \subseteq \hat{E}^*_{crs}$. Informally, this means that DDB scores better with respect to minimality of change then CRS.

8 Conclusion

We have applied the general idea of theory-expansion to logic programs with constraints.
We have shown that sometimes expansion functions are easy to compute but also that they can be rather hard. Especially when we apply *semantically* motivated minimality criteria like minimal change expansion, the problem of finding such expansions is NP-hard.
We can try, however, by applying the tractable *syntactically* motivated minimality criteria to appoximate minimal change expansions. For example among the syntactically minimal expansions, expansion functions based on (maximal) foundations can be used to minimize the model change.
Possible extensions of our research could be

(i) comparing expansion frameworks using different underlying semantics of logic programs, such as the two-valued stable model semantics and extensions of the well-founded semantics.
(ii) studying the expansion framework within restricted programming classes like stratified programs and definite programs.
(iii) investigating the relationship between abduction and program expansion.
(iv) the application of the framework to more general non-monotonic formalisms as default logic and auto-epistemic logic.

References

1. C. Alchourrón, P. Gärdenfors and D. Makinson, On the Logic of Theory Change: Partial Meet Contraction and Revision Functions, *Journal of Symbolic Logic*, **50**, 510–530, 1985.
2. J. Dix, Classifying Semantics of Logic Programs. In: A. Nerode et al. (eds), *Proceedings LPNMR'91*, MIT Press, 1991, pp. 166-180.
3. J. Doyle, A Truth Maintenance System, *Artificial Intelligence* **12**, 1979.
4. P. Gärdenfors, *Knowledge in Flux*, MIT Press, Cambridge, MA, 1988.
5. M. R. Garey, D. S. Johnson, *Computers and Intractability*, Freeman, New York, 1979.
6. G. Gottlob, C. G. Fermüller, Removing redundancy from a clause, *Artificial Intelligence*, **61**, (1993) 263–289
7. C. M. Jonker, Cautious Backtracking and Well-Founded Semantics in Truth Maintenance. Technical Report RUU-CS-91-26, Dept. of Computer Science, Utrecht University, 1991.
8. C. M. Jonker, Analysis of Dependency-Directed Backtracking: leading to Informative Backtracking. Technical Report to appear, Dept. of Philosophy and Dept. of Computer Science, Utrecht University, 1993.
9. Morris, P., Stable Closures, Defeasible Logic and Contradiction Tolerant Reasoning, *Proceedings of the 7th National Conference on Artificial Intelligence*, 1988 .
10. 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.
11. 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.
12. L. M. Pereira, Personal Communication, Berlin 1992.
13. Petrie, C.J., Revised Dependency-Directed Backtracking for Default Reasoning, Proc. AAAI, 1987.
14. H. Przymusinska and T. Przymusinski, Semantic Issues in Deductive Databases and Logic Programs, in: R.B. Banerji (ed), *Formal Techniques in Artificial Intelligence, A Sourcebook*, Elsevier, Amsterdam, 1990, pp. 321-367.
15. T. Przymusinski, Well-founded semantics coincides with three-valued stable semantics, *Fundamenta Informaticae*, XIII:445–463, 1990.
16. T. Przymusinski, Three-valued nonmonotonic formalisms and semantics of logic programs, *Artificial Intelligence*, **49**, (1991), 309–343.
17. Reinfrank, M., Fundamentals and Logical Foundations of Truth Maintenance, Linköping Studies in Science and Technology. Dissertations no. 221, Linköping University, 1989.
18. A. Van Gelder, K. A. Ross and J. S. Schlipf, The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3), pp. 620–650, 1991.
19. K. W. Wagner, Bounded Query Classes, *Siam Journal On Computing*,**19**,5, pp. 833–846, 1990.
20. C. Witteveen, Expansions of Logic Programs, in: D. Pearce and G. Wagner (eds), *Logics in AI*, Springer Verlag, Berlin, 1992.
21. C. Witteveen and C. M. Jonker, Revision by expansion in logic programs, Reports of the Faculty of Technical Mathematics and Informatics no. 93-02, Delft University of Technology, 1993.
22. C. Witteveen and G. Brewka, Skeptical Reason Maintenance and Belief Revision, *Artificial Intelligence*, **61** (1993) 1–36.