Partial Meet Revision and Contraction in Logic Programs

Sebastian Binnewies and **Zhiqiang Zhuang** and **Kewen Wang**
School of Information and Communication Technology
Griffith University, QLD, Australia
{s.binnewies; z.zhuang; k.wang}@griffith.edu.au

Abstract

The recent years have seen several proposals aimed at placing the revision of logic programs within the belief change frameworks established for classical logic. A crucial challenge of this task lies in the nonmonotonicity of standard logic programming semantics. Existing approaches have thus used the monotonic characterisation via SE-models to develop semantic revision operators, which however neglect any syntactic information, or reverted to a syntax-oriented belief base approach altogether. In this paper, we bridge the gap between semantic and syntactic techniques by adapting the idea of a partial meet construction from classical belief change. This type of construction allows us to define new model-based operators for revising as well as contracting logic programs that preserve the syntactic structure of the programs involved. We demonstrate the rationality of our operators by testing them against the classic AGM or alternative belief change postulates adapted to the logic programming setting. We further present an algorithm that reduces the partial meet revision or contraction of a logic program to performing revision or contraction only on the relevant subsets of that program.

Introduction

For three decades now has the study of belief change addressed the dynamics within knowledge representation systems. The most widely-adopted belief change approach is the so-called AGM framework (Alchourrón, Gärdenfors, and Makinson 1985; Gärdenfors 1988), which classifies the possible changes to a knowledge base as *expansion*, *revision*, and *contraction* operations. On the one hand, the framework provides a set of postulates that each rational change operator should satisfy, and, on the other hand, defines specific operators that satisfy these criteria. Although the AGM approach has been applied to a variety of knowledge representation formalisms (Wassermann 2011), its adaptation to logic programs faces a major challenge in form of the nonmonotonicity of standard logic programming semantics. Yet, only recently was belief revision, as understood in the strict sense of the AGM framework, adapted to logic programs in a seminal work (Delgrande et al. 2013). The adaptation rests upon characterising an agent's beliefs in terms of possible worlds, more specifically, in terms of the set of *SE-models* of a logic program. SE-models provide a monotonic characterisation for logic programs and thus circumvent any obstacles presented by nonmonotonicity. For the revision of a program P by a program Q, the set-containment-based operator returns those SE-models from the set of SE-models of Q that are closest to the SE-models of P, denoted by $SE(P \star Q)$. Closeness is determined by computing the set difference between SE-models; we refer to the original paper for further details.

While this adaptation is clearly a milestone in merging classical belief change with a nonmonotonic knowledge representation formalism, we point out some drawbacks of this approach. Specifically, we question whether taking the set of SE-models as the characterisation of beliefs expressed by a program is the correct choice. Before explaining our point of view further, we start by giving some examples using the proposed revision operator. For convenience, we provide a possible program that corresponds to the revision outcome for each example, which we denote by $P \star Q$. In the examples, we assume that the underlying language contains only the symbols that occur in the programs.

1) $P = \{\, a.,\ b \leftarrow a. \,\}$ $\quad Q = \{\, \bot \leftarrow a. \,\}$
   $SE(P \star Q) = \{(b, b)\}$ $\quad P \star Q = \{\, \bot \leftarrow a.,\ b. \,\}$
2) $P = \{\, \bot \leftarrow a.,\ b \leftarrow not\, a. \,\}$ $\quad Q = \{\, a. \,\}$
   $SE(P \star Q) = \{(ab, ab)\}$ $\quad P \star Q = \{\, a.,\ b. \,\}$
3) $P = \{\, a.,\ b \leftarrow not\, c. \,\}$ $\quad Q = \{\, \bot \leftarrow c. \,\}$
   $SE(P \star Q) = \{(ab, ab)\}$ $\quad P \star Q = \{\, a.,\ b.,\ \bot \leftarrow c. \,\}$
4) $P = \{\, a.,\ b \leftarrow not\, a. \,\}$ $\quad Q = \{\, \bot \leftarrow a. \,\}$
   $SE(P \star Q) = \{(\emptyset, \emptyset), (\emptyset, b), (b, b)\}$
   $P \star Q = \{\, \bot \leftarrow a. \,\}$
5) $P = \{\, \bot \leftarrow a.,\ b \leftarrow a. \,\}$ $\quad Q = \{\, a. \,\}$
   $SE(P \star Q) = \{(a, a), (a, ab), (ab, ab)\}$ $\quad P \star Q = \{\, a. \,\}$

In Example 1), the initial belief state expressed by program P consists of a and b. In fact, the second rule in P says that we believe b *if* we believe a. After revising by the program Q, which simply states that we do not believe a, we still believe b even though the reason to believe b is not given anymore. The explanation for this is that the revision operator acts on a program-level, not on a rule-level, as it

considers just the SE-models of the program in its entirety. However, the dependency of b on a is not captured by the SE-models of the program, only by the SE-models of the second rule. Therefore, b is treated as an independent fact during the revision process. The situation is similar in Example 2). Here, we initially believe b due to the absence of belief a. After the revision, we continue to believe b even though the grounds for b do not exist anymore.

In Example 3), we are indifferent with respect to b initially and should believe b when we do not believe c. The revising program Q contains information that c indeed does not hold. Thus, b is incorporated into the new belief state. Yet, Example 4) describes a similar scenario in which b is not included in the resulting belief state, thereby showing a clear discrepancy to the behaviour of the revision operator in the previous example. It seems that some dependencies between atoms expressed in P are respected, while others are not.

Comparing Examples 4) and 5) demonstrates another characteristic of the revision operator. In both examples, the revision operation effectively disregards the second rule in P. This is due to the fact that the set of SE-models of P is exactly the set of SE-models of the first rule. The second rule is thus *locally irrelevant* (Delgrande and Wang 2014). Consequently, the revision operator returns a result as if P had consisted merely of the first rule.

We can attribute these two characteristics of the revision operator to its focus on the program-level. In such an approach, a program may freely be substituted with any other that has the same set of SE-models and the revision output will remain the same. Obviously, this is a direct consequence of the assumption that an agent's beliefs are represented by the set of SE-models of a program. Yet, the information expressed by a program is more than just its set of SE-models – a program also encodes any relationships between the atoms occurring in it (Leite and Pereira 1998). We take these relationships into account in the construction of belief change operators for logic programs presented in the next sections. Instead of a program-level perspective, we take a more fine-grained, *rule-level* perspective by understanding a belief state as the collection of the sets of SE-models of all rules in a program, in the spirit of (Slota and Leite 2012). We represent a belief state in the format of a program, such that each rule in the program represents one element of the collection.

The contributions of this paper are the following:

- We define model-based operators for the expansion, revision, and contraction of logic programs that preserve the syntactic information about relationships between atoms encoded in a program. Our operators thereby bridge the gap between semantic and syntactic belief change approaches, and, to the best of our knowledge, present the first definition of a model-based contraction operator for logic programs.
- We adapt the classic AGM belief change postulates to the logic programming setting and show that our operators satisfy all major postulates. For those postulates that are not satisfied, we discuss their suitability, provide alternative postulates, and demonstrate compliance.
- We introduce the notion of a *module* as a subset of a program and define which modules are relevant to a revision or contraction operation. Using this definition, we develop an algorithm that reduces the revision or contraction of a program to revising or contracting only the relevant modules of that program.

Preliminaries

We first briefly recall syntax and semantics of logic programs (Lifschitz, Pearce, and Valverde 2001; Turner 2003) and review the foundations of belief change.

Logic Programming

Let $\mathcal{A}$ be a finite vocabulary of propositional atoms. A *rule* r over $\mathcal{A}$ has the form

$$a_1; \dots ; a_k; not\, b_1; \dots ; not\, b_l \leftarrow c_1, \dots, c_m, \\ not\, d_1, \dots, not\, d_n. \quad (1)$$

Here, all $a_i, b_i, c_i, d_i \in \mathcal{A}$ and $k, l, m, n \geq 0$. The operators 'not', ';', and ',' stand for default negation, disjunction, and conjunction, respectively. For convenience, let $H^+(r) = \{a_1, \dots, a_k\}$, $H^-(r) = \{b_1, \dots, b_l\}$, $B^+(r) = \{c_1, \dots, c_m\}$, and $B^-(r) = \{d_1, \dots, d_n\}$. If $k = 1$ and $l = m = n = 0$, then r is called a *fact* and we omit '$\leftarrow$'; if $k = l = 0$, then r is a *constraint* and we denote the empty disjunction by $\bot$. Let $At(r)$ and $At(R)$ denote the set of all atoms that occur in a rule of the form (1) and in a set of rules R, respectively. A *(generalised) logic program* is a finite set of rules of the form (1). We write $\mathcal{LP}$ for the class of all logic programs.

An *interpretation* $Y \subseteq \mathcal{A}$ satisfies a program P, denoted by $Y \models P$, iff it is a model of all rules under the standard definition for propositional logic such that each rule represents a standard conditional and default negation is transcribed to classic negation. An *answer set* of a program P is any subset-minimal interpretation Y that satisfies the *reduct* P^Y of P with respect to Y, defined as: $P^Y = \{\, H^+(r) \leftarrow B^+(r) \mid r \in P, H^-(r) \subseteq Y, \text{ and } B^-(r) \cap Y = \emptyset \,\}$.

An *SE-interpretation* is a tuple (X, Y) of interpretations with $X \subseteq Y \subseteq \mathcal{A}$. We usually write, e.g., (ab, ab) instead of $(\{a, b\}, \{a, b\})$ for legibility. Let $\mathcal{SE}$ be the set of all SE-interpretations over $\mathcal{A}$. For any set S of SE-interpretations, by $\overline{S}$ we denote the complement of S with respect to $\mathcal{SE}$, that is, $\overline{S} = \mathcal{SE} \setminus S$. An SE-interpretation (X, Y) is an *SE-model* of a program P iff $Y \models P$ and $X \models P^Y$. The set of all SE-models of P is denoted by $SE(P)$ and P is *satisfiable* iff $SE(P) \neq \emptyset$. For any rule $r \in P$, we obtain the SE-models of r by setting $SE(r) = SE(P')$ for $P' = \{r\}$. Note that $SE(P) = \bigcap_{r \in P} SE(r)$. Given two programs P, Q, we say that P is *strongly equivalent* to Q, denoted by $P \equiv_s Q$, iff $SE(P) = SE(Q)$, and P *implies* Q, denoted by $P \models_s Q$, iff $SE(P) \subseteq SE(Q)$. Furthermore, we write $\models_s P$ to express $SE(P) = \mathcal{SE}$.

Belief Change

The AGM framework (Alchourrón, Gärdenfors, and Makinson 1985; Gärdenfors 1988) defines *expansion*, *revision*, and

contraction as the belief change operations on a knowledge base in light of some new information. In an expansion or revision, the new information is added to the knowledge base, and in the case of revision, existing information that conflicts with the new information is removed. Contraction refers to the process of removing some information from the knowledge base without adding any new information.

We call the information expressed by a knowledge base a *belief state*. In the AGM framework, a belief state is modelled as a *belief set*, defined as a set of sentences from some logic-based language that is closed under logical consequence. Here, we list the formula-based versions (Katsuno and Mendelzon 1991) of the AGM revision postulates, as they are more amenable to an adaptation for logic programs. They are as follows, where $\oplus$ denotes a revision operator and ϕ, χ, ψ are propositional formulas.

($\oplus$1) $\phi \oplus \chi$ implies χ

($\oplus$2) If $\phi \wedge \chi$ is satisfiable, then $\phi \oplus \chi \equiv \phi \wedge \chi$

($\oplus$3) If χ is satisfiable, then $\phi \oplus \chi$ is satisfiable

($\oplus$4) If $\phi_1 \equiv \phi_2$ and $\chi_1 \equiv \chi_2$, then $\phi_1 \oplus \chi_1 \equiv \phi_2 \oplus \chi_2$

The AGM postulates for contraction are given below, where $\ominus$ represents a contraction operator, K is a belief set, ϕ is a formula, and $Cn(\cdot)$ stands for a logical consequence function.

($\ominus$1) $K \ominus \phi = Cn(K \ominus \phi)$

($\ominus$2) $K \ominus \phi \subseteq K$

($\ominus$3) If $\phi \notin K$, then $K \ominus \phi = K$

($\ominus$4) If $\not\vdash \phi$, then $\phi \notin K \ominus \phi$

($\ominus$5) $K \subseteq Cn\left((K \ominus \phi) \cup \{\phi\}\right)$

($\ominus$6) If $\phi_1 \equiv \phi_2$, then $K \ominus \phi_1 = K \ominus \phi_2$

A New Approach to Belief Change under SE-Models

Before we develop the construction of partial meet revision and contraction in the next two sections, we start by defining the *expansion* of a logic program. In the following sections, we use P and Q to denote two arbitrary logic programs.

Definition 1. *Let P be a logic program. An operator $+$ is an* expansion operator *for P such that, for any logic program Q,*

$$P + Q = P \cup Q.$$

This definition is a direct adaptation of the original formulation of expansion (Alchourrón, Gärdenfors, and Makinson 1985): to obtain the expansion of P by Q, all rules of Q are added to those of P, even if the outcome will be an inconsistent program.

Partial Meet Revision

The adaptation of the AGM revision postulates to logic programs is straightforward and given below, where $*$ is a function from $\mathcal{LP} \times \mathcal{LP}$ to $\mathcal{LP}$:

($*$1) $P * Q \models_s Q$

($*$2) If $P + Q$ is satisfiable, then $P * Q \equiv_s P + Q$

($*$3) If Q is satisfiable, then $P * Q$ is satisfiable

($*$4) If $P_1 \equiv_s P_2$ and $Q_1 \equiv_s Q_2$, then $P_1 * Q_1 \equiv_s P_2 * Q_2$

We now generate a partial meet construction of logic program revision. As the basis for our construction, we define a *compatible set* of some program with respect to another program as the dual of a remainder set (Alchourrón, Gärdenfors, and Makinson 1985).

Definition 2. *Let P, Q be logic programs. The set of* compatible sets *of P with respect to Q is*

$$\mathbb{P}_Q = \{\, R \mid R \subseteq P,\ SE(R) \cap SE(Q) \neq \emptyset,\ \textit{and for all } R' \textit{ with } R \subset R' \subseteq P : SE(R') \cap SE(Q) = \emptyset \,\}.$$

Each compatible set is a maximal subset of P that is consistent with Q under SE semantics. Each is thus a candidate to be returned together with Q as the outcome of a revision. To determine exactly which candidate(s) to choose, we employ a *selection function*.

Definition 3. *A* selection function γ *for a set M is a function such that:*

1. $\gamma(M) \subseteq M$;
2. *if $M \neq \emptyset$, then* $\gamma(M) \neq \emptyset$.

We can now define *partial meet revision* for logic programs as the intersection of the selected compatible sets added to Q.

Definition 4. *Let P be a logic program. For any logic program Q, we define an operator $*$ as a* partial meet revision operator *for P such that*

$$P * Q = \bigcap \gamma(\mathbb{P}_Q) + Q.$$

Example 1. *Let $P = \{\, a.,\ b \leftarrow a. \,\}$ and $Q = \{\, \bot \leftarrow a. \,\}$. We have $\mathbb{P}_Q = \{\, \{\, b \leftarrow a. \,\} \,\} = \gamma(\mathbb{P}_Q)$, for any selection function γ, and thus $P * Q = \{\, b \leftarrow a.,\ \bot \leftarrow a. \,\}$.*

The following theorem states that the revision operator satisfies all major postulates from above.

Theorem 1. *The revision operator $*$ satisfies ($*$1), ($*$2), and ($*$3).*

Proof. ($*$1): Since $Q \subseteq P * Q$, it holds that $P * Q \models_s Q$.
($*$2): If $P + Q$ is satisfiable, then $\mathbb{P}_Q = \{P\} = \gamma(\mathbb{P}_Q)$, for any selection function γ, and thus $P * Q = P + Q$.
($*$3): Let Q be satisfiable. For any $R \in \mathbb{P}_Q$, $R + Q$ is satisfiable, which implies $P * Q$ is satisfiable. □

Postulate ($*$4) is called *Syntax-Independence* and requires that the SE-models of a revision outcome remain the same if we substitute the initial and the revising program each by strongly equivalent programs. However, as pointed out above, the information content of a logic program is captured not only by its set of SE-models, but also by its rules that encode the relationships between atoms in the program. Therefore, postulate ($*$4) is too strict a requirement for any logic program revision operator that respects semantic as well as syntactic information. The operator $*$ is such an operator and thus does not satisfy ($*$4), as shown in the next example.

Example 2. *Let* $P' = \{\, a.,\ b. \,\}$ *and* $Q = \{\, \bot \leftarrow a. \,\}$. *We have* $\mathbb{P}'_Q = \{\, \{\, b. \,\} \,\} = \gamma(\mathbb{P}_Q)$, *for any selection function* γ, *and thus* $P * Q = \{\, b.,\ \bot \leftarrow a. \,\}$.

Although P from Example 1 and P' from Example 2 are strongly equivalent, we can see that $P * Q$ and $P' * Q$ are not, because $SE(P * Q) = \{(\emptyset,\emptyset),(\emptyset,b),(b,b)\} \neq \{(b,b)\} = SE(P' * Q)$. Alternatively, we consider the weakening of ($*4$), also known as *Weak Independence of Syntax* (Osorio and Cuevas 2007):

($*4$w) If $Q_1 \equiv_s Q_2$, then $P * Q_1 \equiv_s P * Q_2$.

The next proposition then follows directly from Definition 4.

Proposition 1. *The revision operator* $*$ *satisfies* ($*4$w).

Returning to our motivation, we can see that partial meet revision addresses the shortcomings of the distance-based revision method. For the examples in the introduction we obtain:

1) $SE(P * Q) = \{(\emptyset,\emptyset),(\emptyset,b),(b,b)\}$
   $P * Q = \{\, \bot \leftarrow a.,\ b \leftarrow a. \,\}$
2) $SE(P * Q) = \{(a,a),(a,ab),(ab,ab)\}$
   $P * Q = \{\, a.,\ b \leftarrow not\ a. \,\}$
3) $SE(P * Q) = \{(ab,ab)\}$
   $P * Q = \{\, a.,\ b \leftarrow not\ c.,\ \bot \leftarrow c. \,\}$
4) $SE(P * Q) = \{(b,b)\}$
   $P * Q = \{\, \bot \leftarrow a.,\ b \leftarrow not\ a. \,\}$
5) $SE(P * Q) = \{(ab,ab)\}$ $\quad P * Q = \{\, a.,\ b \leftarrow a. \,\}$

In Examples 1) and 2), the partial meet revision operator preserves the dependency of b on a and $not\ a$, respectively. This is expressed on the syntactic level by the revised program $P * Q$ and on the semantic level by $SE(P * Q)$. Regarding Examples 3) and 4), the partial meet revision operator treats the dependency of b on $not\ c$ and $not\ a$, respectively, in the same manner and adds b to the belief state uniformly in both examples. Finally, the partial meet revision operator takes into account all rules in a program, even those that may be "invisible" from a purely model-based perspective, as shown by the outcomes for Examples 4) and 5).

Partial Meet Contraction

Having defined a revision operator, we now turn to the case of belief contraction. The following is an adaptation of the AGM contraction postulates, where $\dot{-}$ is a function from $\mathcal{LP} \times \mathcal{LP}$ to $\mathcal{LP}$:

($\dot{-}1$) $P \dot{-} Q$ is a logic program

($\dot{-}2$) $P \dot{-} Q \subseteq P$

($\dot{-}3$) If $P \not\models_s Q$, then $P \dot{-} Q = P$

($\dot{-}4$) If $\not\models_s Q$, then $P \dot{-} Q \not\models_s Q$

($\dot{-}5$) $(P \dot{-} Q) + Q \models_s P$

($\dot{-}6$) If $P_1 \equiv_s P_2$ and $Q_1 \equiv_s Q_2$, then $P \dot{-} Q_1 \equiv_s P \dot{-} Q_2$

Again, we focus on individual rules of a program and their models as the basis of our construction. In line with classic belief change, the contraction of a program P by a program Q should eliminate from P all those beliefs from which Q can be derived. We use the complement of $SE(Q)$, $\overline{SE(Q)}$, to determine all subsets of P that do not imply Q, denoted as

$$\mathbb{P}^-_Q = \{\, R \mid R \subseteq P,\ SE(R) \cap \overline{SE(Q)} \neq \emptyset,\ \text{and for all } R' \text{ with } R \subset R' \subseteq P : SE(R') \cap \overline{SE(Q)} = \emptyset \,\}.$$

Definition 5. *Let* P *be a logic program. For any logic program* Q, *we define an operator* $*$ *as a* partial meet contraction operator *for* P *such that*

$$P \dot{-} Q = \bigcap \gamma(\mathbb{P}^-_Q).$$

Example 3. *Let* $P = \{\, a.,\ b \leftarrow a. \,\}$ *and* $Q = \{\, a \leftarrow b. \,\}$. *Since* $SE(\{\, a. \,\}) = \{(a,a),(a,ab),(ab,ab)\}$, $SE(\{\, b \leftarrow a. \,\}) = \{(\emptyset,\emptyset),(\emptyset,b),(b,b),(\emptyset,ab),(b,ab),(ab,ab)\}$, *and* $\overline{SE(Q)} = \{(\emptyset,b),(b,b),(b,ab)\}$, *we have* $\mathbb{P}^-_Q = \{\, \{\, b \leftarrow a. \,\} \,\} = \gamma(\mathbb{P}^-_Q)$, *for any selection function* γ, *and thus* $P \dot{-} Q = \{\, b \leftarrow a. \,\}$.

The contraction operator complies with all major postulates given above.

Theorem 2. *The contraction operator* $\dot{-}$ *satisfies* ($\dot{-}1$), ($\dot{-}2$), ($\dot{-}3$), *and* ($\dot{-}4$).

Proof. ($\dot{-}1$) and ($\dot{-}2$): Follow directly from Definition 5.
($\dot{-}3$): If $P \not\models_s Q$, then $\mathbb{P}^-_Q = \{P\} = \gamma(\mathbb{P}^-_Q)$, for any selection function γ, and thus $P \dot{-} Q = P$.
($\dot{-}4$): Let $\not\models_s Q$. For any $R \in \mathbb{P}^-_Q$, $R \not\models_s Q$, which implies $P \dot{-} Q \not\models_s Q$. □

The *Recovery* postulate ($\dot{-}5$) states that a contraction operator should not retract information unduly from the original belief state. The operator $\dot{-}$ does not satisfy ($\dot{-}5$) because a contracted program $P \dot{-} Q$ may share SE-models with Q that are not part of the set of SE-models of the initial program P. In fact, the adequacy of postulate ($\ominus 5$) for belief sets under classical logic has been disputed intensively, and several alternative postulates exist that capture the minimal change property in its place. Here, we adapt the *Relevance* postulate (Hansson 1989) and show that $\dot{-}$ complies with it:

($\dot{-}5$r) If $r \in P$ and $r \notin P \dot{-} Q$, then there exists a program P' such that $P \dot{-} Q \subseteq P' \subseteq P$ and $P' \not\models_s Q$ but $P' \cup \{r\} \models_s Q$.

Proposition 2. *The contraction operator* $\dot{-}$ *satisfies* ($\dot{-}5$r).

Proof. Let $r \in P$. Assume that for all P' with $P \dot{-} Q \subseteq P' \subseteq P$ and $P' \not\models_s Q$, it holds that $P' \cup \{r\} \not\models_s Q$. In particular, for each $R' \in \mathbb{P}^-_Q$ with $P \dot{-} Q \subseteq R'$, this implies $R' \cup \{r\} \not\models_s Q$. As each R' is subset-maximal, it follows that $r \in R'$ and thus $r \in P \dot{-} Q$. □

As in the case of revision, the Syntax-Independence postulate ($\dot{-}6$) is too strong and instead we consider its weakened version:

($\dot{-}6$w) If $Q_1 \equiv_s Q_2$, then $P \dot{-} Q_1 \equiv_s P \dot{-} Q_2$.

The next proposition follows directly from Definition 5.

Proposition 3. *The contraction operator* $\dot{-}$ *satisfies* ($\dot{-}6$w).

Localised Belief Change

The formation of a set of compatible sets requires that all possible combinations of all rules in a program are evaluated with respect to their sets of SE-models. When dealing with logic programs that contain a large number of rules, where only a small number of them are actually affected by the change operation, this procedure entails unreasonable costs. In this section, we present an algorithm to minimise these costs. We begin by identifying the subsets of a program, called *modules*, relevant to another program.

Definition 6. *Let P be a logic program and $a \in \mathcal{A}$. For any rule $r \in P$ with $a \in At(r)$, we recursively construct $M(P)^r_i|_a$ as*

$$r \cup \{\, r' \in P \mid At(r') \cap \big(At(r) \cup At(M(P)^r_{i-1}|_a)\big) \setminus a \neq \emptyset \,\}$$

for $i > 0$ and $M(P)^r_0|_a = \emptyset$.

Since P is finite and $M(P)^r_i|_a$ is monotonic with respect to i, the sequence $\bigcup_{i=0}^{\infty} M(P)^r_i|_a$ will reach a fixpoint. We denote the fixpoint by $M(P)^r|_a$ and call it the module of P related to r including a *(or the r-*module including a*, if P is clear from the context).*

Example 4. *Let r_1: $a.$, r_2: $b \leftarrow a.$, r_3: $c \leftarrow not\, b.$, and $P = \{r_1, r_2, r_3\}$. The modules that can be constructed from P are: $M(P)^{r_1}|_a = \{r_1\}, M(P)^{r_2}|_a = \{r_2, r_3\}, M(P)^{r_2}|_b = \{r_1, r_2\}, M(P)^{r_3}|_b = \{r_3\}$, and $M(P)^{r_3}|_c = \{r_1, r_2, r_3\}$.*

Starting with a given atom a and a given rule r from P, the recursive definition first finds all rules in P that share atoms with r except for a. Then it finds all rules in P that share atoms with r or any of the rules found in the first step except for a, and so on. It does not matter whether atoms appear in the head or the body of a rule, or whether they occur with or without default negation. The resulting module is the collection of rules in P that are related to r through shared atoms. The reason for excluding a will become clear after the following definition of a set of modules.

Definition 7. *Let P be a logic program. Given an atom $a \in \mathcal{A}$, we define the* set of all modules of P including a *as:*

$$\mathcal{M}(P)|_a = \{\, M(P)^r|_a \mid r \in P \text{ and } a \in At(r) \,\}.$$

Given a logic program Q, we define the set of all modules of P relevant to Q *as:*

$$\mathcal{M}(P)|_Q = \{\, M(P)^r|_a \mid r \in P \text{ and } a \in At(r) \cap At(Q) \,\}.$$

Essentially, the definition of a set of modules extracts those rules from a program that may be affected during a revision or contraction by another program. It thus aims for the same goal as the language-splitting technique in propositional logic (Parikh 1999), which splits a knowledge base into several partitions either relevant or irrelevant to a belief change. However, a distinct feature in the previous definitions is the construction of a module based on *each* rule in which a certain atom occurs. This feature allows us to get a closer look at which rules may conflict with some given information. Consider the program $\{\, a \leftarrow b., \bot \leftarrow b. \,\}$. If we were to add the information that "b holds" to this program, it would conflict with the latter rule but not with the first one. By creating a module for each occurrence of b, we split the program into two modules (one for each rule) and can assess the compatibility of each module with the new information separately. Furthermore, by constructing modules for each individual atom occurring in Q, this ensures that we are dealing with minimal units of P in a change operation. Obviously, a module may not be unique to a certain rule or a certain given atom so that modules may overlap or coincide.

We say that a set of rules R *conflicts* with a program Q if $SE(R) \cap SE(Q) = \emptyset$. All rules of P that conflict with Q are included in some module or combination of modules from $\mathcal{M}(P)|_Q$.

Proposition 4. *Let P, Q be satisfiable logic programs. For any $R \subseteq P$, if $SE(R) \cap SE(Q) = \emptyset$ and $\forall R' \subset R : SE(R') \cap SE(Q) \neq \emptyset$, then $\exists \mathbb{M} \in 2^{\mathcal{M}(P)|_Q}$ such that $R \subseteq \bigcup \mathbb{M}$.*

Proof. Let P, Q be satisfiable logic programs and $R \subseteq P$ such that $SE(R) \cap SE(Q) = \emptyset$ and for each $R' \subset R : SE(R') \cap SE(Q) \neq \emptyset$. Then there exists some $a_j \in \mathcal{A}$ such that $a_j \in At(Q)$ and there exist one or more rules $r_i \in R$ for each a_j such that $a_j \in At(r_i)$. For each r_i, there exists a corresponding r_i-module $M(P)^{r_i}|_{a_j}$ including a_j, such that $r_i \in M(P)^{r_i}|_{a_j}$. It follows from Definition 6 that for all remaining rules $r' \in R \setminus r_i : r' \in \bigcup_{i,j} M(P)^{r_i}|_{a_j}$. □

Corollary 1. *Let P, Q be logic programs and P be satisfiable. Then $SE(P) \cap SE(Q) = \emptyset$ if and only if $SE\left(\bigcup \mathcal{M}(P)|_Q\right) \cap SE(Q) = \emptyset$.*

Proof. "If": Since $SE(P) \subseteq SE\left(\bigcup \mathcal{M}(P)|_Q\right)$, if $SE\left(\bigcup \mathcal{M}(P)|_Q\right) \cap SE(Q) = \emptyset$, then also $SE(P) \cap SE(Q) = \emptyset$.
"Only if": Follows from Proposition 4 if Q is satisfiable. Trivial if Q is not satisfiable. □

Algorithm 1: MODCHANGE

Input: a set $\mathcal{M}$ of modules, an operator $\circ$, a program Q
Output: the set $\mathcal{M}$ of changed modules

```
n ← 1;
while n ≤ |M| do
    foreach M ⊆ M such that |M| = n do
        if ⋃M ∘ Q ≠ ⋃M then
            foreach M ∈ M do
                replace M with ⋃M ∘ Q in M;
            end
        end
    end
    n ← n + 1;
end
return M;
```

Algorithm 1 resolves potential conflicts for all possible combinations of modules by employing the partial meet revision and contraction operators defined above. It performs a bottom-up construction by first taking all 1-combinations

(singleton sets of modules) of $\mathcal{M}$ and substituting a module with its changed version if they are not the same. It then takes all 2-combinations of $\mathcal{M}$, which may now contain some changed modules, and replaces each module of the combination with the changed version of the combination if required. Replacing *each* module of a combination with the outcome guarantees that the algorithm considers all possible combinations. The algorithm terminates after handling the combination of all modules in $\mathcal{M}$.

The next theorem states that, given any selection function γ, the algorithm MODCHANGE reduces a partial meet revision or contraction operation on a logic program to the revision or contraction operation on the relevant subsets of that program.

Theorem 3. *For any two logic programs P, Q, let $P \setminus \mathcal{M}(P)|_Q = \{ r \in P \mid \forall M \in \mathcal{M}(P)|_Q : r \notin M \}$ and $\mathcal{M}(P)|_Q^\circ$ denote the output of Algorithm 1 for the inputs $\mathcal{M}(P)|_Q$, $\circ \in \{*, \dot{-}\}$, and Q. Then $P * Q = P \setminus \mathcal{M}(P)|_Q + \bigcup \mathcal{M}(P)|_Q^* + Q$ (or $P \dot{-} Q = P \setminus \mathcal{M}(P)|_Q + \bigcup \mathcal{M}(P)|_Q^{\dot{-}}$, respectively) for some selection function γ over $\mathbb{P}_Q$ ($\mathbb{P}_Q^-$, respectively).*

Proof. To prove the equation for revision, we need to show that $P \setminus \mathcal{M}(P)|_Q + \bigcup \mathcal{M}(P)|_Q^* = \bigcap \gamma(\mathbb{P}_Q)$ for some γ. Let $Z \subseteq P$ be the set of rules that are eliminated during the revision operation, i.e., $P * Q = \bigcap \gamma(\mathbb{P}_Q) + Q = P \setminus Z + Q$, and let $Z' \subseteq P$ be the set of rules that are eliminated by MODCHANGE.

We first show that $Z \subseteq Z'$. Assume that $Z' = \emptyset$ until the last iteration of the while-loop. In the last iteration, we have $n = |\mathcal{M}(P)|_Q|$ and MODCHANGE computes $\bigcup \mathcal{M}(P)|_Q * Q = \bigcup \mathcal{M}(P)|_Q^*$. Since $At(P \setminus \mathcal{M}(P)|_Q) \cap At(\bigcup \mathcal{M}(P)|_Q) = \emptyset$, it holds that $P \setminus \mathcal{M}(P)|_Q + \bigcup \mathcal{M}(P)|_Q^* = P \setminus \mathcal{M}(P)|_Q + (\bigcup \mathcal{M}(P)|_Q * Q) = ((P \setminus \mathcal{M}(P)|_Q) \cup \bigcup \mathcal{M}(P)|_Q) * Q = P * Q$ for some γ over $\mathbb{P}_Q$.

We now show that $Z' \subseteq Z$. Assume that each revision operation in the following is the most restrictive type, that is, for any set M, $\gamma(M) = M$. Thus, if $r \in \bigcap \gamma(\mathbb{P}_Q)$, then $r \in R$ for all $R \in \mathbb{P}_Q$. For each $\mathbb{M}$ as specified in Line 3 of MODCHANGE, let z' be the set of rules eliminated during the revision of $\bigcup \mathbb{M}$ by Q: $z' = \bigcup \mathbb{M} \setminus (\bigcup \mathbb{M} * Q)$. From $SE(P) \subseteq SE(\bigcup \mathbb{M})$ it then follows that $z' \cap \bigcap \gamma(\mathbb{P}_Q) = \emptyset$. Consequently, $Z' \cap \bigcap \gamma(\mathbb{P}_Q) = \emptyset$.

Analogous for contraction. □

Discussion

In this paper, we have presented expansion, revision, and contraction operators to execute belief change actions on logic programs. By using the monotonic SE-model semantics and the notion of a compatible set, we were able to directly adapt a partial meet construction from classical logic. The advantage of our approach is that it performs belief change on the semantic level with respect to the SE-models of programs, while at the same time preserving the syntactic information represented by their rules. Furthermore, we showed that the operators exhibit desirable properties. They satisfy nearly the entire set of adapted postulates. As intended, the Syntax-Independence postulate does not hold but its weakened version does. In the case of the highly-disputed Recovery postulate, we adapted a well-established alternative and showed compliance. Finally, we developed a module-based algorithm that prunes a partial meet revision or contraction operation to performing the change only on the relevant subsets of a program.

Our work was motivated by identifying some shortcomings of the distance-based approach to logic program revision (Delgrande et al. 2013). We showed that our partial meet revision operator addresses these drawbacks and possesses similar properties regarding the AGM rationality postulates. Additionally, the choice of a partial meet construction based on compatible sets facilitated a natural definition of model-based contraction for logic programs, the first in this field to the best of our knowledge.

Following the distance-based construction of logic program revision reviewed above, representation theorems have been given (Delgrande, Peppas, and Woltran 2013; Schwind and Inoue 2013), which state that any logic program revision operator satisfying the adapted set of AGM postulates can be suitably characterised by some preorder over a set of SE-interpretations. It is left for future work to show that the partial meet revision operator defined here can be characterised in terms of such a preorder.

A more syntax-oriented proposal is the *belief base* approach to logic program revision (Krümpelmann and Kern-Isberner 2012), which understands a belief state as a set of rules. Its revision operation first finds all subsets of a program that are consistent with the revising program under answer set semantics, and then employs a selection function over these subsets, similar to our revision method. However, to guarantee satisfiability of the revision outcome, the selection function is restricted to choosing exactly one subset. The *program-level* approach to revision (Delgrande 2010) is also based on answer set semantics and as such faces similar obstacles. It makes use of three-valued answer sets during the revision operation to satisfy a core subset of the AGM postulates.

The landscape of logic program "update" operators has been reviewed exhaustively, for a detailed overview see (Slota 2012). Of interest here is the *exception-based update* approach (Slota and Leite 2012), which regards a program as the collection of the sets of *RE-models* of its rules, an extension of SE-models. In the update operation, exceptions in the form of RE-models are added to all sets of RE-models of the initial program that are incompatible with respect to the RE-models of the updating program. In contrast to our method, this approach is a purely semantic one and determines incompatibilities between two sets of RE-models by finding differences in the truth values of atoms occurring in both sets.

Regarding classical logics, Nebel (1991) introduced *base revision* which takes syntactic information into consideration as a measure for revising beliefs in a minimal way. As it is a fully syntax-based operation, however, revising two knowledge bases that represent the same belief state with the same information may lead to different resulting belief states. Partial meet constructions have so far been adapted predominantly to monotonic non-classical for-

malisms (Wassermann 2011), such as the Horn fragment of propositional logic (Delgrande and Wassermann 2013; Zhuang and Pagnucco 2011) or Description Logics (Ribeiro and Wassermann 2009). It would thus be of interest to investigate the adaptation to extensions of logic programs, such as hybrid knowledge bases (Binnewies et al. 2013), as well as to other nonmonotonic formalisms.

Given that for logic programs under answer set semantics revision can be characterised by forgetting (Eiter and Wang 2008), we plan to look at the relationship between our revision operation and SE forgetting (Delgrande and Wang 2015) in future work. Additionally, a comparison with further constructions for logic program revision and contraction, e.g., epistemic entrenchment, may provide useful insights.

Acknowledgments

We thank the anonymous referees for their valuable comments. This work was partially supported by the Australian Research Council under DP110101042 and DP130102302.

References

Alchourrón, C. E.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic* 50(2):510–530.

Binnewies, S.; Wang, Y.; Stantic, B.; and Wang, K. 2013. Rule revision in Normal DL Logic Programs. In *Web Reasoning and Rule Systems*, volume 7994 of *Lecture Notes in Computer Science*. 204–209.

Delgrande, J. P., and Wang, K. 2014. An approach to forgetting in disjunctive logic programs that preserves strong equivalence. In *Proceedings of the 15th International Workshop on Non-Monotonic Reasoning (NMR 2014)*, 38–44.

Delgrande, J. P., and Wang, K. 2015. A syntax-independent approach to forgetting in disjunctive logic programs. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015*. In press.

Delgrande, J. P., and Wassermann, R. 2013. Horn clause contraction functions. *Journal of Artificial Intelligence Research* 48(1):475–511.

Delgrande, J. P.; Schaub, T.; Tompits, H.; and Woltran, S. 2013. A model-theoretic approach to belief change in answer set programming. *ACM Transactions on Computational Logic* 14(2):14:1–14:46.

Delgrande, J. P.; Peppas, P.; and Woltran, S. 2013. AGM-style belief revision of logic programs under answer set semantics. In *Logic Programming and Nonmonotonic Reasoning*, volume 8148 of *Lecture Notes in Computer Science*. 264–276.

Delgrande, J. P. 2010. A program-level approach to revising logic programs under the answer set semantics. *Theory and Practice of Logic Programming* 10(Special Issue 4–6):565–580.

Eiter, T., and Wang, K. 2008. Semantic forgetting in answer set programming. *Artificial Intelligence* 172(14):1644–1672.

Gärdenfors, P. 1988. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press.

Hansson, S. O. 1989. New operators for theory change. *Theoria* 55(2):114–132.

Katsuno, H., and Mendelzon, A. O. 1991. Propositional knowledge base revision and minimal change. *Artificial Intelligence* 52(3):263–294.

Krümpelmann, P., and Kern-Isberner, G. 2012. Belief base change operations for answer set programming. In *Logics in Artificial Intelligence*, volume 7519 of *Lecture Notes in Computer Science*. 294–306.

Leite, J. A., and Pereira, L. M. 1998. Generalizing updates: From models to programs. In *Logic Programming and Knowledge Representation*, volume 1471 of *Lecture Notes in Computer Science*. 224–246.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic* 2(4):526–541.

Nebel, B. 1991. Belief revision and default reasoning: Syntax-based approaches. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, 417–428.

Osorio, M., and Cuevas, V. 2007. Updates in answer set programming: An approach based on basic structural properties. *Theory and Practice of Logic Programming* 7(4):451–479.

Parikh, R. 1999. Beliefs, belief revision, and splitting languages. *Logic, Language and Computation* 2:266–278.

Ribeiro, M. M., and Wassermann, R. 2009. Base revision for ontology debugging. *Journal of Logic and Computation* 19(5):721–743.

Schwind, N., and Inoue, K. 2013. Characterization theorems for revision of logic programs. In *Logic Programming and Nonmonotonic Reasoning*, volume 8148 of *Lecture Notes in Computer Science*. 485–498.

Slota, M., and Leite, J. 2012. Robust equivalence models for semantic updates of answer-set programs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012*.

Slota, M. 2012. *Updates of hybrid knowledge bases*. Ph.D. Dissertation, Universidade Nova de Lisboa.

Turner, H. 2003. Strong equivalence made easy: Nested expressions and weight constraints. *Theory and Practice of Logic Programming* 3(4):609–622.

Wassermann, R. 2011. On AGM for non-classical logics. *Journal of Philosophical Logic* 40(2):271–294.

Zhuang, Z., and Pagnucco, M. 2011. Transitively relational partial meet Horn contraction. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, 1132–1138.