Towards Causality-Based Conflict Resolution in Answer Set Programs

Andre Thevapalan(✉), Konstantin Haupt, and Gabriele Kern-Isberner

Technische Universität Dortmund, 44227 Dortmund, Germany
{andre.thevapalan,konstantin.haupt,
gabriele.kern-isberner}@cs.tu-dortmund.de

Abstract. Using answer set programming in real-world applications requires that the answer set program is correct and adequately represents knowledge. In this paper, we present strategies to resolve unintended contradictory statements resulting from modelling gaps and other flaws by modifying the program without manipulating the actual conflicting rules (inconsistency-causing rules with complementary head literals). We show how *latent conflicts* can be detected to prevent further conflicts during the resolution process or after subsequent modifications in the future. The presented approach is another step towards a general framework where professional experts who are not necessarily familiar with ASP can repair existing answer set programs and independently resolve conflicts resulting from contradictory statements in an informative way. In such a framework, conflict resolution strategies allow for generating possible solutions that consist of informative extensions and modifications of the program. In interaction with the professional expert, these solution options can then be used to obtain the solution that represents the underlying knowledge best.

Keywords: Answer Set Programming · Conflicts · Consistency · Contradictions · Interactive Conflict Resolution

1 Introduction

In order to use answer set programming in real-world applications, the utilized answer set programs must be modelled correctly and represent professionally adequate knowledge. Especially in large programs, unintended contradictory statements due to modelling gaps and other flaws are hard to detect, and repairing a knowledge base can require both a professional expert and a technical expert. Approaches as in [1,2] rewrite an updated program using the *causal rejection principle* [2] such that in case of contradictory derivations, newer knowledge is preferred. These approaches, however, are actually "workarounds" that change the program in an automated fashion and do not guarantee that the underlying cause of the inconsistency is eliminated. In practice, such a solution may not always be desirable, e. g. when inconsistencies hint at outdated information or modelling flaws. Instead, one may wish to resolve the actual conflicting statements in the program. In [10], it is shown how rules directly involved in a *conflict*

G. Gottlob et al. (Eds.): LPNMR 2022, LNAI 13416, pp. 350–362, 2022.
https://doi.org/10.1007/978-3-031-15707-3_27

can be modified to achieve a consistent program. In this paper, we extend this approach by showing how conflicts can be resolved by modifying rules other than the conflicting ones in case the conflicting rules themselves are thought to be adequate. For that, we first outline the general relations between a conflict rule and the rest of the program. Based on these results, we exemplarily showcase two different strategies that utilize the different rule dependencies to prevent conflicting rules from being simultaneously satisfiable. In place of exploiting technical devices like causal rejection, the presented strategies, as in [10], produce informative solution options from which a professional expert can then choose the most suitable one, where informative means that modifications of rules make use of the given program language. Another aspect we cover is *latent conflicts*. We show that even consistent programs can contain contradictory knowledge that "hides" behind consistent answer sets due to the nature of answer set semantics that chooses only consistent answer sets, ignoring the inconsistent fixpoints. Accordingly, such latent conflicts should also be resolved in the process to achieve robust consistency. This way, it is guaranteed that every implemented modification to establish consistency conclusively leads to a program that adequately reflects professional knowledge.

After presenting the necessary theoretical preliminaries in Sect. 2, we introduce the notion of latent conflicts and causality-based conflict resolution in Sect. 3 that allows us to resolve conflicts without manipulating the conflicting rules themselves. In Sect. 4, we examine the general conditions for conflicting rules to become simultaneously derivable. Using these results and the definitions of the prior section, we present two exemplary strategies that follow the causality-based conflict resolution approach. After presenting a brief overview over related work in Sect. 5, the paper concludes with Sect. 6, giving a short summary and discussion of open issues and further work.

2 Preliminaries

In this paper, we look at non-disjunctive *extended logic programs* (ELPs) [6]. An ELP is a finite set of rules over a set $\mathcal{A}$ of propositional atoms. A (classical) literal L is either an atom A (*positive literal*) or a negated atom $\neg A$ (*negative literal*). For a literal L, the *strongly complementary* literal $\overline{L}$ is $\neg A$ if $L = A$ and A otherwise. A *default-negated literal* L, called *default literal*, is written as $\sim L$. A set S of literals is *consistent* iff S does not contain strongly complementary literals. A *rule* r is of the form $L_0 \leftarrow L_1, \ldots, L_m, \sim L_{m+1}, \ldots, \sim L_n.$, with literals $L_0, \ldots, L_n$ and $0 \leq m \leq n$. The literal L_0 is the *head* of r, denoted by $H(r)$, and $\{L_1, \ldots L_m, \sim L_{m+1}, \ldots \sim L_n\}$ is the *body* of r, denoted by $B(r)$. Furthermore, $\{L_1, \ldots, L_m\}$ is denoted by $B^+(r)$ and $\{L_{m+1}, \ldots, L_n\}$ by $B^-(r)$. A rule r with $B(r) = \emptyset$ is called a *fact*, and r is called a *constraint* if it has an empty head. An *extended logic program (ELP)* $\mathcal{P}$ is a set of rules.

Given a set S of literals, we say S *satisfies* L iff $L \in S$, and S *satisfies* $\sim L$ iff $L \notin S$. S *satisfies a rule body* $B(r)$ iff for all $L \in B^+(r)$, S satisfies L, and for all $L \in B^-(r)$, S satisfies $\sim L$. S *satisfies a rule* r iff S satisfies $H(r)$ whenever S

satisfies $B(r)$. In case r is a constraint, S satisfies r iff S does not satisfy $B(r)$. A rule body $B(r)$ is *satisfiable* if there exists a set S of literals such that S satisfies $B(r)$. Given a set $\mathcal{R}$ of rules, S *satisfies* $\mathcal{R}$ if S satisfies all $r \in \mathcal{R}$ simultaneously. S is a *pre-model of* $\mathcal{P}$ if S satisfies $\mathcal{P}$. A pre-model S of $\mathcal{P}$ is a *model of* $\mathcal{P}$ if S is consistent. A rule r is *active under* S iff S satisfies $B(r)$. The set of all rules in $\mathcal{P}$ that are active under S is denoted by $act^S(\mathcal{P})$. Given a literal L, H_L denotes the set of rules in $\mathcal{P}$ with the rule head L, i. e. $H_L = \{r \in \mathcal{P} \mid H(r) = L\}$. Given two rules $r, r' \in \mathcal{P}$, r is *supported (resp. opposed) by* r' iff there exists a literal $L \in B^+(r)$ (resp. $L \in B^-(r)$) such that $L = H(r')$. Then, r' is a *supporting (resp. opposing) rule of* r.

In the following, we extend the basic idea of answer sets to inconsistent sets of literals. Given an ELP $\mathcal{P}$ without default negation, the *pre-answer set* of $\mathcal{P}$ is a set S of literals such that S satisfies $\mathcal{P}$ and S is $\subseteq$-minimal. In general, an *answer set* of an ELP $\mathcal{P}$ is determined by its reduct. The *reduct* $\mathcal{P}^S$ of a program $\mathcal{P}$ relative to a set S of literals is defined by

$$\mathcal{P}^S = \{H(r) \leftarrow B^+(r). \mid r \in \mathcal{P}, B^-(r) \cap S = \emptyset\}.$$

A set S of literals is a *pre-answer set* of $\mathcal{P}$ if it is the pre-answer set of $\mathcal{P}^S$ [6]. A pre-answer set S is a *(classical) answer set* of $\mathcal{P}$ if S is consistent, otherwise, we call S a *pseudo answer set*. The set of all pseudo answer sets of a program $\mathcal{P}$ is denoted by $AS^{\perp}(\mathcal{P})$, the set of all (classical) answer set by $AS(\mathcal{P})$, and the set of all pre-answer sets $AS^{\perp}(\mathcal{P}) \cup AS(\mathcal{P})$ by $AS^{pre}(\mathcal{P})$. Note that $AS(\mathcal{P})$ aligns with the set of answer sets under usual definitions, e. g. [6].

For every literal of a pre-answer set S, there must exist a rule $r \in \mathcal{P}$ with $H(r) = L$ s. t. $r \in act^S(\mathcal{P})$. This is in complete analogue to answer sets [4].

Example 1. Let $\mathcal{P}_1$ be the ELP of our running example in this paper:

r_1: $drugA \leftarrow condW.$ $\quad$ r_2: $\overline{drugA} \leftarrow sympT.$

r_3: $treatmZ \leftarrow drugC, {\sim}drugD.$ $\quad$ r_4: $\overline{treatmZ} \leftarrow sympQ, {\sim}drugD, {\sim}sympR.$

r_5: $condW \leftarrow sympU.$ $\quad$ r_6: $drugC \leftarrow {\sim}drugD, {\sim}sympR.$

r_7: $drugD \leftarrow {\sim}drugC.$ $\quad$ r_8: $sympQ.$ r_9: $sympT.$ r_{10}: $sympU.$

$\mathcal{P}_1$ models a knowledge base regarding the conditions for prescribing a drug A and a treatment Z, as well as the conditions when they must not be prescribed. For example, r_3 represents the condition that treatment Z may only be recommended if the patient is already taking drug C and drug D is not known to be applied, while r_4 says that patients must not be treated with Z if they currently show symptom Q, drug D is not known to be applied, and symptom R does not seem to be developed.

$\mathcal{P}_1$ has the pre-answer sets $S_1 = \{condW, drugA, \overline{drugA}, drugD, sympQ, sympT, sympU\}$ and $S_2 = \{condW, drugA, \overline{drugA}, drugC, sympQ, sympT, sympU, treatmZ, \overline{treatmZ}\}$. Both answer sets are pseudo answer sets as they contain complementary literals. □

To formalize modification operations on programs, we introduce suitable modification operations:

Definition 1 (Rule Modification Operation). *Let $\mathcal{P}$ be an ELP with a rule $r \in \mathcal{P}$ and X a set of literals and default literals where either $X \subseteq B(r)$ or $X \cap B(r) = \emptyset$. We define the* rule modification operation *$rmod(r, X)$ as follows:*

$$rmod(r, X) = \begin{cases} H(r) \leftarrow B(r) \backslash X & \text{if } X \subseteq B(r) \\ H(r) \leftarrow B(r) \cup X & \text{if } X \cap B(r) = \emptyset \end{cases}$$

In the following, a *program modification operation* $\mathcal{P} \diamond r^\star$ is either the addition of a new rule $r^\star$ to $\mathcal{P}$, i. e. $\mathcal{P} \cup \{r^\star\}$ or the replacement of a rule $r \in \mathcal{P}$ by a modification $r^\star = rmod(r, X)$ of r, i. e. $\mathcal{P} \backslash \{r\} \cup \{r^\star\}$. We express a set of modifications that shall be applied to $\mathcal{P}$ by simply listing the new rules and rule modifications, i. e. $R^\star = \{r_1^\star, \ldots, r_n^\star\}$ denotes a set of modification elements, where $r_i^\star$ $(1 \leq i \leq n)$ is either a new rule for $\mathcal{P}$ or a modification $rmod(r, X)$ with $r \in \mathcal{P}$ and a set of (default) literals X. Given a set $R^\star = \{r_1^\star, \ldots, r_n^\star\}$ of rule modifications and new rules for $\mathcal{P}$, the consecutive application of each element onto $\mathcal{P}$ is denoted by $\mathcal{P} \diamond R^\star$, i. e. $\mathcal{P} \diamond R^\star = (((\mathcal{P} \diamond r_1^\star) \diamond r_2^\star) \diamond \ldots \diamond r_n^\star)$.

Example 2 (Example 1 contd.). Suppose studies have shown that condition W always presents with a symptom V. The addition of a literal $sympV$ to r_5 can then be written as $rmod(r_5, \{sympV\})$, resulting in a rule $r_5^\star$: $condW \leftarrow sympU, sympV$. Replacing r_5 in $\mathcal{P}_1$ by $r_5^\star$ can then be expressed by $\mathcal{P}_1 \diamond rmod(r_5, \{sympV\})$. □

3 Causality-Based Conflict Resolution

In this section, we introduce the notion of *derivable conflicts* and outline a method named *causality-based conflict resolution* that can be used to describe how to modify an inconsistent program to achieve consistency. During the resolution process, for every derivable conflict, a set of modifications is applied to $\mathcal{P}$. Every set of changes leads to the resolution of the respective conflict where the modifications are built from elements of the underlying language of $\mathcal{P}$.

3.1 Conflicts and Inconsistency

In this paper, we show strategies to resolve inconsistencies that are caused by rules with complementary head literals by constructing informative program modifications. To that end, we first specify the type of inconsistency that our strategies aim to resolve.

Definition 2 (Consistency, Contradictory Program). *An ELP $\mathcal{P}$ is called* consistent *iff $AS(\mathcal{P}) \neq \emptyset$. $\mathcal{P}$ is* contradictory *iff $AS(\mathcal{P}) = \emptyset$ and $AS^\perp(\mathcal{P}) \neq \emptyset$.*

This paper solely deals with programs that are inconsistent due to contradictions.

Example 3 (Example 1 contd.). Program $\mathcal{P}_1$ is inconsistent as the two existing pre-answer sets are pseudo answer sets. □

Definition 3 (Conflicting Rules, Conflict (cf. [10])). *Two rules r_1, r_2 in an ELP $\mathcal{P}$ are* conflicting *if $H(r_1)$ and $H(r_2)$ are strongly complementary and there exists a consistent set S of literals such that $B(r_1)$ and $B(r_2)$ are satisfied by S. A* conflict *is a set $\mathcal{C} = \{r_1, r_2\}$ of rules such that r_1 and r_2 are conflicting. We denote the set of all conflicts $\{r_1, r_2\}$ in an ELP $\mathcal{P}$ by $Conflicts(\mathcal{P})$.*

A program that contains conflicts is not necessarily inconsistent. A conflict $\{r_1, r_2\}$ in $\mathcal{P}$ is only a potential cause of inconsistency whenever the body literals of both $B(r_1)$ and $B(r_2)$ can be simultaneously derived in $\mathcal{P}$. The program's pseudo answer sets can be used to determine whether the body literals of conflicting rules are derivable and which rules are causing inconsistency.

Definition 4 (Derivably Conflicting Rules). *Given an ELP $\mathcal{P}$ over $\mathcal{A}$, a conflict $\{r_1, r_2\} \in Conflicts(\mathcal{P})$ is* derivable[1] *in $\mathcal{P}$ iff there exists a pseudo answer set $S \in AS^{\perp}(\mathcal{P})$ s. t. r_1 and r_2 are active under S. Otherwise, a conflict is* nonderivable. *The set of all derivable conflicts in $\mathcal{P}$ is denoted by $Conflicts^{dv}(\mathcal{P})$.*

However, a conflict is only (co-)responsible for inconsistency if the conflict is derivable. Thus, a program is consistent if it does not possess derivable conflicts.

Proposition 1. *Let $\mathcal{P}$ be an ELP with $Conflicts(\mathcal{P}) \neq \emptyset$. $\mathcal{P}$ is consistent if every conflict in $Conflicts(\mathcal{P})$ is nonderivable, i. e. $Conflicts^{dv}(\mathcal{P}) = \emptyset$.*

Proof. Let $\mathcal{P}$ be an ELP with $Conflicts(\mathcal{P}) \neq \emptyset$ and $Conflicts^{dv}(\mathcal{P}) = \emptyset$. Suppose $\mathcal{P}$ is inconsistent. Then there exists a pseudo answer set $S \in AS^{\perp}(\mathcal{P})$, which in turn implies that there exist two conflicting rules r_1 and r_2 that are active under S. By definition, $\{r_1, r_2\}$ is a derivable conflict which contradicts our initial assumption that $Conflicts^{dv}(\mathcal{P}) = \emptyset$. □

Example 4 (Example 1 contd.). $\mathcal{P}_1$ has two conflicts $\mathcal{C}_1 = \{r_1, r_2\}$ and $\mathcal{C}_2 = \{r_3, r_4\}$. Its answer sets show that both conflicts are derivable since the complementary literals in the answer sets can only originate from the rules in $\mathcal{C}_1$ and $\mathcal{C}_2$. Answer set S_1 shows that to achieve a consistent program, it suffices to make either $drugA$ or $\overline{drugA}$ nonderivable as r_1 and r_2 are the only conflicting rules that are active under S_1. Let $\mathcal{P}_4$ be $\mathcal{P}_1$ where the body of r_5 is extended by a literal $sympV$, i. e. $\mathcal{P}_4 = \mathcal{P}_1 \diamond rmod(r_5, \{sympV\})$. $\mathcal{P}_4$ now has the pre-answer sets $S_{4,1} = \{\overline{drugA}, drugD, sympQ, sympT, sympU\}$, and $S_{4,2} = \{\overline{drugA}, drugC, sympQ, sympT, sympU, treatmZ, \overline{treatmZ}\}$. Since $S_{4,1}$ is not a pseudo answer set, $\mathcal{P}_4$ is consistent. □

Example 4 depicts the relationship between derivable conflicts and consistency.

Depending on the actual program modifications, further inconsistency-causing conflicts, so called *latent conflicts*, can be revealed. These conflicts should therefore be handled correspondingly to ensure that after resolving all conflicts, the program is indeed consistent.

[1] Note that literals are classified as derivable once they appear in a pre-answer set and not only in a (classical) answer set.

Definition 5 (Latent Conflict). *Let $\mathcal{P}$ be an ELP with a conflict $\mathcal{C}$. $\mathcal{C}$ is a* latent conflict *if $\mathcal{C}$ is a derivable conflict and $AS(\mathcal{P}) \neq \emptyset$.*

Example 5 (Example 4 contd.). In $\mathcal{P}_4$, conflict $\mathcal{C}_2 = \{r_3, r_4\}$ is a latent conflict as its rules are active under $S_{4,2}$ but not in the (consistent) answer $S_{4,1}$. Suppose that it was prescribed that in order to take drug D, the patient also has to show symptom V. Thus, program $\mathcal{P}_4$ has to be modified to a program $\mathcal{P}_5$ by adding the literal $sympV$ to the body of r_7, i. e. $\mathcal{P}_5 = \mathcal{P}_4 \diamond rmod(r_7, \{sympV\})$. $\mathcal{P}_5$ is now inconsistent as its only pre-answer set is the pseudo answer set $S_{4,2}$ from Example 4 in which the rules of $\mathcal{C}_2$ are active. □

Given the goal of solely "repairing" an inconsistent knowledge base, modifying the corresponding inconsistent program until one gets at least one answer set seems appropriate (see Example 4). Considering Example 5, one can easily imagine that in the process of resolving conflicts, other previously latent conflicts can become effective, meaning they can cause inconsistency.

3.2 Conflict Resolution

In this paper, we want to present strategies to explicitly "resolve" all derivable conflicts (which include latent conflicts) in a knowledge base in an informative way, that is, every change made to obtain a consistent program is based on the underlying language. Therefore, each such strategy modifies the program so that every (derivable) conflict becomes nonderivable, which as a consequence also ensures the resolution of latent conflicts. We now outline, given a derivable conflict $\mathcal{C}$ in an ELP $\mathcal{P}$, how one can modify $\mathcal{P}$ to a program $\mathcal{P}^\star$ such that $\mathcal{C}$ becomes nonderivable in $\mathcal{P}^\star$ without manipulating the conflicting rules themselves. We call this approach *causality-based conflict resolution*. There, in each step, a conflict is resolved by applying suitable program modifications to $\mathcal{P}$. The resolution process results in a modified program $\mathcal{P}^\star$ that is free of derivable conflicts and thereby consistent.

Definition 6 (Causality-Based Conflict Resolution Step and Process). *Let $\mathcal{P}$ be an ELP with $Conflicts^{dv}(\mathcal{P}) \neq \emptyset$. Given a conflict $\mathcal{C} = \{r_1, r_2\} \in Conflicts^{dv}(\mathcal{P})$, a* causality-based conflict resolution step *in $\mathcal{P}$ w.r.t. $\mathcal{C}$ is the modification of $\mathcal{P}$ to $\mathcal{P}^\star$ such that r_1 and r_2 are not derivably conflicting in $\mathcal{P}^\star$. A* causality-based conflict resolution process *w.r.t. $\mathcal{P}$ is a sequence $\langle \mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$ where $\mathcal{P}_0 = \mathcal{P}$ and for each $\mathcal{P}_i, \mathcal{P}_{i+1} (0 \leq i < n)$, $\mathcal{P}_{i+1}$ is the result of a causality-based conflict resolution step in $\mathcal{P}_i$, and $\mathcal{P}_n$ contains no derivable conflicts, i. e. $Conflicts^{dv}(\mathcal{P}_n) = \emptyset$.*

The following example illustrates a causality-based resolution process consisting of a single conflict resolution step.

Example 6 (Example 5 contd.). Let $\mathcal{P}_6$ be the following program that extends $\mathcal{P}_5$ from Example 5 by the following two additional rules:

$$r_{11}\colon drugC \leftarrow sympT, {\sim}drugD. \qquad r_{12}\colon drugC \leftarrow \overline{sympQ}, sympR.$$

Here, conflict $\mathcal{C}_2 = \{r_3, r_4\}$ is the only derivable conflict, hence a single conflict resolution step suffices. One possible set of modifications is $R^\star = \{r_6^\star, r_{11}^\star\}$ with $r_6^\star = rmod(r_6, \{\sim sympT\})$ and $r_{11}^\star = rmod(r_{11}, \{sympV\})$. Program $\mathcal{P}_6 \diamond R^\star$ has the unique answer set $\{\overline{drugA}, sympQ, sympT, sympU, \overline{treatmZ}\}$. □

Ergo, in each conflict resolution step, a derivable conflict becomes nonderivable after applying a suitable set of modifications to the program. We now present two concrete strategies to build suitable program modifications for a conflict.

4 Strategies for Conflict Resolution

After describing the general properties that have to be satisfied in order for two rules to be not simultaneously satisfiable, we present two explicit strategies to resolve derivable conflicts. While in [10], the conflicting rules themselves are modified during a conflict resolution step, the demonstrated strategies yield conflict-preventing sets that do not involve the conflicting rules by using causality-based conflict resolution. As in [10], the inconsistent program is modified using informative extensions, i. e. the modifications are based on the underlying language rather than inventing technical workarounds. The following section presents some technical considerations and results that will prove useful for the strategies.

4.1 General Satisfaction Interdependencies

We propose strategies for manipulating $\mathcal{P}$ to a consistent program $\mathcal{P}^\star$ such that $AS^\perp(\mathcal{P}^\star) = \emptyset$ where for every conflict $\mathcal{C} = \{r_1, r_2\}$ in $\mathcal{P}^\star$ and $S \in AS^{pre}(\mathcal{P}^\star)$, it holds that if S satisfies $B(r_1)$, then S does not satisfy $B(r_2)$. In other words, whenever r_1 is active under a pre-answer set, r_2 cannot become active.

Proposition 2. *Let $\mathcal{P}$ be an ELP with $Conflicts(\mathcal{P}) \neq \emptyset$. $\mathcal{P}$ is consistent if there exists at least one pre-answer set S in $\mathcal{P}$ such that for every conflict $\mathcal{C} = \{r_1, r_2\} \in Conflicts(\mathcal{P})$, the following holds:*

$$S \text{ satisfies } B(r_1) \implies S \text{ does not satisfy } B(r_2) \tag{1}$$

Proof. Let S be a pre-answer set such that (1) holds for all conflicts $\mathcal{C} = \{r_1, r_2\}$ in $Conflicts(\mathcal{P})$. Assume S is inconsistent. Then there exists at least one conflict $\mathcal{C} \in Conflicts(\mathcal{P})$ such that S satisfies both $B(r_1)$ and $B(r_2)$, which contradicts the initial specification of S. Thus, every pre-answer set of $\mathcal{P}$ where (1) holds for every conflict in $Conflicts(\mathcal{P})$ is an answer set. □

Regarding the non-satisfiability of a body, we derive the following assertion:

Proposition 3. *Given a pre-answer set S of an ELP $\mathcal{P}$ and a rule $r \in \mathcal{P}$, the following holds (where* sat. *stands for satisfies/satisfy):*

$$\begin{aligned} S \text{ not sat. } B(r) \quad & \text{iff } \exists L \in B^-(r) \text{ s.t. } L \in S \text{ or} \\ & \quad \exists L \in B^+(r) \text{ s.t. } L \notin S, \end{aligned} \tag{2}$$

$$\begin{aligned} & \text{iff } \exists r' \in H_L, L \in B^-(r) \text{ s.t. } S \text{ sat. } B(r') \text{ or} \\ & \quad \forall r' \in H_L, L \in B^+(r) \text{ s.t. } S \text{ not sat. } B(r'). \end{aligned} \tag{3}$$

Proof. Let S be a pre-answer set for an ELP $\mathcal{P}$ and $r \in \mathcal{P}$ a rule in $\mathcal{P}$. Equation (2) follows directly from the definition of the satisfaction of rule bodies in Sect. 2. Equation (3) can be shown by using the properties of pre-answer sets: Since pre-answer sets are defined as minimal sets of literals that satisfy all rules in a program, each contained literal has to be derived by at least one rule whose body is satisfied and therefore adds its head literal to the pre-answer set. If there is at least one rule $r' \in H_L$ such that $L \in B^-(r)$ and S satisfies $B(r')$, the head literal $H(r') = L$ has to be contained in S. Therefore, (2) is fulfilled, and $B(r)$ is not satisfied by S. Analogously, if for every rule $r' \in H_L$ with $L \in B^+(r)$, S does not satisfy $B(r')$, L cannot be contained in S due to the minimality of pre-answer sets. Again, (2) demands that $B(r)$ is not satisfied by S. □

The interdependencies illustrated in Proposition 3 can be used to define different strategies to modify a program such that a conflict rule becomes nonderivable without changing the conflicting rules. Basically, (3) demands that any set of literals that satisfies $B(r_2)$ for a conflict $\mathcal{C} = \{r_1, r_2\}$ should satisfy an opposing rule of r_1 or that there exists a positive body literal $L \in B^+(r_1)$ such that every supporting rule of r_1 w.r.t. to L is not satisfied by S. Proposition 3 also illustrates the recursive nature of (non-)satisfiability of rules in logic programs, as the second case in (3) implies (2) with r replaced by each r'.

The proposed strategies focus on manipulating the answer sets in such a way that the conflicting rules can still be active if the other conflicting rule is not active under an answer set. The proposed strategies can therefore be seen as a switch that prevents a conflicting rule from becoming active whenever the other conflict rule is active under an answer set. Thereby, the approach in Sect. 4.2 exploits the rule dependencies of negative body literals while the alternative strategy in Sect. 4.3 exploits the rule dependencies of positive body literals.

4.2 Blocking Rules Using Opposing Rules

Our first approach to prevent the derivability of conflicting rules are so-called *blocking rules*. A blocking rule adds a specific literal to each pseudo answer set in which one of the conflicting rules is active. In particular, this specific literal is part of the negative body of the other conflicting rule, which therefore gets "blocked". Thus, no pre-answer set can contain the complementary (head) literals of the conflicting rules at the same time. As a blocking rule exploits the negative body literals of a conflicting rule, this approach is only applicable for conflicts of the form $\mathcal{C} = \{r_1, r_2\}$ with $B^-(r_1) \backslash B^-(r_2) \neq \emptyset$.

Definition 7 (Blocking Rule). *Let $\mathcal{P}$ be an ELP with a conflict $\mathcal{C} = \{r_1, r_2\}$ for which $B^-(r_1) \backslash B^-(r_2) \neq \emptyset$ holds. Then, a* blocking rule *$r^\star$ for $\mathcal{C}$ is a rule*

$$r^\star\colon E \leftarrow B(r_2)., \textit{ where } E \in B^-(r_1) \backslash B^-(r_2).$$

Definition 7 demands that the bodies of $r^\star$ and r_2 are identical. As the goal is to assure that $r^\star$ becomes active under a set of literals whenever r_2 is active, it would suffice that $B(r^\star) \subseteq B(r_2)$ holds. This, however, can lead to unwanted

side effects as $r^\star$ can become active in cases where r_2 is not, which then again could lead to additional (latent) conflicts.

Proposition 4. *Let $\mathcal{P}$ be an ELP with a derivable conflict $\mathcal{C} = \{r_1, r_2\}$ and $r^\star$ a blocking rule for $\mathcal{C}$. Then, it holds that $\mathcal{C}$ becomes nonderivable in $\mathcal{P} \diamond r^\star = \mathcal{P} \cup r^\star$, i. e. for every pre-answer set $S \in AS^{pre}(\mathcal{P} \diamond r^\star)$, $r_1, r_2 \notin act^S(\mathcal{P} \diamond r^\star)$ is true.*

Proof. Let $\mathcal{C} = \{r_1, r_2\}$ be a derivable conflict in $\mathcal{P}$ s. t. $B^-(r_1) \backslash B^-(r_2) \neq \emptyset$ and $r^\star$ a blocking rule for $\mathcal{C}$ which is meant to block r_1, i. e. $H(r^\star) \in B^-(r_1) \backslash B^-(r_2)$ and $B(r^\star) = B(r_2)$. If $\mathcal{C}$ was derivable in $\mathcal{P}^\star = \mathcal{P} \diamond r^\star$, there would have to exist a pseudo answer set $S \in AS^\perp(\mathcal{P}^\star)$ with $r_1, r_2 \in act^S(\mathcal{P}^\star)$. As $r_2 \in act^S(\mathcal{P}^\star)$ is assumed, $r^\star \in act^S(\mathcal{P}^\star)$ holds (due to $B(r^\star) = B(r_2)$). Thus, $H(r^\star) \in S$ is true. But, as $H(r^\star) \in B^-(r_1)$, r_1 cannot be active under S, which contradicts the assumption of r_1 and r_2 being simultaneously active under S. Thus, there cannot be any pseudo answer set $S \in AS^\perp(\mathcal{P}^\star)$ under which r_1 and r_2 are active. As due to the definition of answer sets, r_1 and r_2 can also not be active under any answer set of $AS(\mathcal{P}^\star)$, $\mathcal{C}$ is nonderivable in all pre-answer sets of $\mathcal{P}^\star$. □

Example 7 (Example 6 contd.). To make the conflict $\mathcal{C}_2 = \{r_3, r_4\}$ in $\mathcal{P}_6$ nonderivable, the blocking rule $r_{13}^\star$: $sympR \leftarrow drugC, {\sim}drugD.$ can be constructed according to Proposition 4. The head literal of $r_{13}^\star$ is referring to the negative body literal $sympR \in B^-(r_4) \backslash B^-(r_3)$. Because the body of the blocking rule $r_{13}^\star$ equals the body of r_3, $r_{13}^\star$ is guaranteed to be active whenever r_3 is active. The blocking rule $r_{13}^\star$ leads to $\mathcal{C}_2$ being nonderivable in the program $\mathcal{P}_6 \diamond r_{13}^\star$ (Proposition 4). Note that no blocking rule can be constructed for r_3 because $B^-(r_3) \backslash B^-(r_4) = \emptyset$. □

Example 7 implies that there can exist multiple possible solutions for a single conflict. In a corresponding framework, the professional expert has the possibility to interactively determine the most suitable solution for each conflict. This step is also reflected in Algorithm 1, where the complete strategy is summarized.

Algorithm 1: Blocking rules using opposing rules

Input: Program $\mathcal{P}$, Conflict $\mathcal{C} = \{r_1, r_2\} \in Conflicts^{dv}(\mathcal{P})$
Output: Modified program $\mathcal{P}^\star$ with $\mathcal{C} \notin Conflicts^{dv}(\mathcal{P}^\star)$ or $\emptyset$

```
Initialize R* := ∅;
foreach E ∈ B⁻(r1)\B⁻(r2) do
    R* := R* ∪ {E ← B(r2).};
foreach E ∈ B⁻(r2)\B⁻(r1) do
    R* := R* ∪ {E ← B(r1).};
if R* = ∅ then return ∅;                    /* no resolution found */
R* := ChooseSuggestion(R*) ;        /* expert chooses suitable R* ∈ R* */
return P ◇ R*;
```

4.3 Relevant Rule Modification

Instead of adding new rules such as the blocking rules presented in Proposition 4, which represent new opposing rules for conflicting rules, another possibility to resolve conflicts would be to modify the existing supporting rules of conflicting rules. We use the notion of rule modification as presented in [10], but instead of modifying the conflicting rules directly, we focus on the rules that are primarily responsible for the bodies of the conflicting rules to be true in the pseudo answer sets of $\mathcal{P}$. Within the rules H_L w.r.t. a literal $L \in B^+(r)$ of a rule $r \in \mathcal{P}$, we can distinguish between rules that are *relevant* for the conflict resolution, i. e. rules that can potentially be simultaneously active with both conflicting rules and *irrelevant* rules, which cannot become simultaneously active with at least one conflicting rule.

Definition 8 (Relevant Rule). *Let $\mathcal{P}$ be an ELP with a conflict $\mathcal{C} = \{r_1, r_2\}$ and H_L with $L \in B^+(r_1) \cup B^+(r_2)$. A rule $r' \in H_L$ is* relevant *for the conflict resolution of $\mathcal{C}$ if there exists a consistent set of literals S with $r_1, r_2, r' \in act^S(\mathcal{P})$. Otherwise, r' is* irrelevant *for the resolution of $\mathcal{C}$.*

Example 8 (Example 6 contd.). In $\mathcal{P}_6$, r_{12} is not relevant for the resolution of $\mathcal{C}_2 = \{r_3, r_4\}$. As $B^+(r_4) \cup B^+(r_{12}) = \{sympQ, \overline{sympQ}, sympR\}$ contains complementary literals and $B^-(r_4) \cap B^+(r_{12}) \neq \emptyset$ (as well as $B^-(r_3) \cap B^+(r_{12}) \neq \emptyset$), the three rules cannot be satisfied simultaneously by a consistent set of literals. Therefore, r_{12} cannot be responsible for the derivation of *drugC* when the conflicting rules are simultaneously satisfied in a pseudo answer set. □

The following definition extends the modification used in [10] and guarantees that at least one body literal of a conflicting rule is not satisfied if the other conflicting rule is active under a pre-answer set, i. e. every relevant rule of the body literal is modified in a way so that it becomes irrelevant for conflict resolution. The application of this method is illustrated in the ensuing example.

Definition 9 (Relevant rule modification). *Let $\mathcal{C} = \{r_1, r_2\}$ be a conflict in an ELP $\mathcal{P}$ s. t. $B^+(r_1) \backslash B^+(r_2) \neq \emptyset$ holds. Furthermore, suppose an arbitrary literal $L \in B^+(r_1) \backslash B^+(r_2)$ and a rule $r' \in H_L$. Then, $Pot(\mathbf{M}_{r'}) = \{M \mid M \subseteq \mathbf{M}_{r'}\}$ with $\mathbf{M}_{r'} = B(r_2) \backslash B(r')$ implies the powerset of possible modifications for r'. For each $M_{r'} \in Pot(\mathbf{M}_{r'})$ and $\widetilde{M}_{r'} = \{\sim a \mid a \in M_{r'}\}$, each rule modification $r^\star$ of r' of the form $rmod(r', \widetilde{M}_{r'})$ defines a relevant rule modification of r', i. e.*

$$r^\star\colon H(r') \leftarrow B(r'), \widetilde{M}_{r'}.$$

Example 9 (Example 6 contd.). If conflict $\mathcal{C}_2$ in $\mathcal{P}_6$ should be resolved in such a way that r_4 is preferred over r_3, we can manipulate the relevant rules of literals in $B^+(r_3) \backslash B^+(r_4)$. As *drugC* is the only literal in $B^+(r_3) \backslash B^+(r_4)$, the rules $H_{drugC} = \{r_6, r_{11}, r_{12}\}$ have to be considered. As shown in Example 8, rule r_{12} is not relevant for the resolution of $\mathcal{C}_2$, so that only r_6 and r_{11} have to be

modified. According to Definition 9, the following relevant rule modifications can be constructed for r_6 and r_{11}:

$$\begin{aligned} r_6^\star\colon\ & drugC \leftarrow {\sim}drugD, {\sim}sympR, {\sim}sympQ. \\ r_{11,1}^\star\colon\ & drugC \leftarrow sympT, {\sim}drugD, {\sim}sympQ. \\ r_{11,2}^\star\colon\ & drugC \leftarrow sympR, sympT, {\sim}drugD. \\ r_{11,3}^\star\colon\ & drugC \leftarrow sympR, sympT, {\sim}drugD, {\sim}sympQ. \end{aligned}$$ □

Since we require every rule in $\mathcal{P}$ to be satisfiable, any set $M_{r'}$ is also satisfiable. Furthermore, by definition, $M_{r'}$ has no common literal with $B(r')$. Thus, for any set $\widetilde{M}_{r'}$, it holds that $B(r') \cup \widetilde{M}_{r'}$ is also satisfiable.

Proposition 5. *Let $\mathcal{P}$ be an ELP with a derivable conflict $\mathcal{C} = \{r_1, r_2\}$ and $B^+(r_1) \backslash B^+(r_2) \neq \emptyset$. After applying relevant rule modifications $R^\star = \{r_1^\star, \ldots, r_n^\star\}$ for all relevant rules $r_1', \ldots, r_n'$ of a set H_L with $L \in B^+(r_1) \backslash B^+(r_2)$ as defined in Definition 9, $\mathcal{C}$ becomes nonderivable in the resulting program $\mathcal{P}^\star = \mathcal{P} \diamond R^\star = ((\mathcal{P} \diamond rmod(r_1', \widetilde{M}_{r_1'})) \diamond \ldots \diamond rmod(r_n', \widetilde{M}_{r_n'}))$.*

Proof. Let $\mathcal{P}$ be an ELP with a conflict $\mathcal{C} = \{r_1, r_2\}$ and $B^+(r_1) \backslash B^+(r_2) \neq \emptyset$ as well as $L \in B^+(r_1) \backslash B^+(r_2)$. Furthermore, let $\mathcal{P}^\star$ be a program that results from relevant rule modifications of every relevant rule for $\mathcal{C}$. The proof of this proposition is done by contradiction: If $\mathcal{C}$ would still be derivable in $\mathcal{P}^\star$, there would have to be at least one pseudo answer set $S \in AS^\bot(\mathcal{P}^\star)$ with $r_1, r_2 \in act^S(\mathcal{P}^\star)$. Then, by the definition of active rules, S satisfies both $B^+(r_1)$ and $B^+(r_2)$ and L in particular since $L \in B^+(r_1) \backslash B^+(r_2)$. Consequently, there has to be at least one rule $r_i \in act^S(\mathcal{P}^\star)$ with $H(r_i) = L$, which is, by definition, also included in H_L. If this rule is not relevant for the conflict resolution, r_i was not modified but could not be active under S in the first place as we already assume $r_1, r_2 \in act^S(\mathcal{P}^\star)$. If r_i is relevant, it was transformed to a modified rule $r_i^\star$ as shown in Definition 9. Then $B^+(r_i^\star) \cap B^-(r_2) \neq \emptyset$ or $B^-(r_i^\star) \cap B^+(r_2) \neq \emptyset$ holds. In either case, $r_i^\star$ cannot be active under S (i. e. $r_i^\star \notin act^S(\mathcal{P}^\star)$). This contradicts the assumption $r_i^\star \in act^S(\mathcal{P}^\star)$. As there cannot be any active rule $r_i \in H_L$ under S, S does not satisfy $B^+(r_1)$, hence $\mathcal{C}$ has to be nonderivable in $\mathcal{P}^\star$. □

Algorithm 2 summarizes the relevant rule modification strategy. The application of this strategy is illustrated in Example 10.

Example 10 (Example 9 contd.). By extending the bodies of all relevant rules of H_{drugC} with ${\sim}sympQ$, $sympR$, or both, all rules in H_{drugC} are now irrelevant for the conflict resolution of $\mathcal{C}_2$, and it is ensured that $drugC$ cannot be derived in any answer set in which r_4 is active. Therefore, r_3 and r_4 cannot become active simultaneously in any answer set. As each combination of modifications, viz. $R_1^\star = \{r_6^\star, r_{11,1}^\star\}$, $R_2^\star = \{r_6^\star, r_{11,2}^\star\}$, and $R_3^\star = \{r_6^\star, r_{11,3}^\star\}$, constitute a possible way to resolve $\mathcal{C}_2$, $\mathcal{P}_6 \diamond R_1^\star$, $\mathcal{P}_6 \diamond R_2^\star$ and $\mathcal{P}_6 \diamond R_3^\star$ describe three different resolution possibilities for $\mathcal{C}_2$ that use relevant rule modifications. □

Algorithm 2: Relevant rule modification

Input: Program $\mathcal{P}$, Conflict $\mathcal{C} = \{r_1, r_2\} \in \mathit{Conflicts}^{dv}(\mathcal{P})$
Output: Modified program $\mathcal{P}^\star$ with $\mathcal{C} \notin \mathit{Conflicts}^{dv}(\mathcal{P}^\star)$ or $\emptyset$

Initialize $\mathcal{R}^\star := \emptyset$; $X := (B^+(r_1) \backslash B^+(r_2)) \cup (B^+(r_2) \backslash B^+(r_1))$;
foreach $L \in X$ **do**
  $\mathcal{Y} := \emptyset$;
  foreach *relevant rule* $r'_i \in H_L$ **do**
    $Y_i :=$ all possible modifications $\mathit{rmod}(r'_i, \widetilde{M}_{r'_i})$;
    $\mathcal{Y} := \mathcal{Y} \cup Y_i$;
  $\mathcal{R}^\star := \mathcal{R}^\star \cup \{\{r_1^\star, \ldots, r_{|\mathcal{Y}|}^\star\} \mid r_i^\star \in Y_i, 1 \leq i \leq |\mathcal{Y}|\}$
if $\mathcal{R}^\star = \emptyset$ **then return** $\emptyset$; `/* no resolution found */`
$R^\star := \mathit{ChooseSuggestion}(\mathcal{R}^\star)$; `/* expert chooses suitable` $R^\star \in \mathcal{R}^\star$ `*/`
return $\mathcal{P} \diamond R^\star$;

5 Related Work

The presented approach is related to methods developed and investigated in the area of *ASP debugging*. Essentially, debugging approaches as in [5,8] aim to modify knowledge bases of any (not necessarily inconsistent) logic programs in order to remedy a mismatch between the actual semantics of the program and the semantics intended by the modeller. In general, the ability to identify errors in a given program and compute suggestions crucially depends on information by the expert that is given on top of the original program. Alternatively, with the approaches in [3,7], the expert can analyze the program step by step in order to detect error causes. Our approach, however, focuses on a specific subclass of erronous programs where the original program is by itself sufficient to identify the problem and generate suitable solution suggestions. Once possible solutions are available, both the presented method as well as debugging approaches like those based on the meta-programming technique [5] can be used to successively obtain the most suitable solution in interaction with the user.

6 Conclusion and Future Work

We have shown how consistency in an inconsistent program can be achieved without modifying the de facto conflicting rules to improve the usability of answer set programs in practice. The presented causality-based resolution approach is obligatory if the conflicting rules themselves must not be altered. By examining the dependencies between rules, we have shown how the satisfaction of conflicting rules can be prevented. For that, we defined two strategies which can be used to generate informative solutions for a professional expert. The expert can choose the most suitable solution for each conflict to achieve consistency. We have also introduced the notion of latent conflicts that can cause inconsistency either during the conflict resolution process or after subsequent modifications.

The recursive nature of rule dependencies allows for the application of the strategies not only to supporting rules of the conflicting rules, but also to their supporting rules (*transitive conflict resolution*). That leads to even more possibilities to repair a program, such that even small programs can lead to a vast amount of possible solutions for a professional expert to scan through. It is therefore critical that a framework as proposed in [9,10] incorporates different workflows and procedures to efficiently find the most fitting solution for each conflict. Such workflows imply suitable interactions with the professional expert to gather relevant background information with the goal to reduce the amount of solutions which in turn can reduce the overall complexity of such conflict resolution approaches.

In future work, we want to extend the presented approach to cover transitive conflict resolution using established methods from argumentation theory. Moreover, we want to develop methods to resolve multiple conflicts simultaneously.

References

1. Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.C.: Dynamic logic programming. In: Freire-Nistal, J.L., Falaschi, M., Ferro, M.V. (eds.) 1998 Joint Conference on Declarative Programming, APPIA-GULP-PRODE 1998, A Coruña, Spain, 20–23 July 1998, pp. 393–408 (1998)
2. Eiter, T., Fink, M., Sabbatini, G., Tompits, H.: On properties of update sequences based on causal rejection. Theory Pract. Log. Program. **2**, 711–767 (2002)
3. Fichte, J.K., Gaggl, S.A., Rusovac, D.: Rushing and strolling among answer sets - navigation made easy. CoRR abs/2112.07596 (2021)
4. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers (2012)
5. Gebser, M., Pührer, J., Schaub, T., Tompits, H.: A meta-programming technique for debugging answer-set programs. In: Fox, D., Gomes, C.P. (eds.) Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, 13–17 July 2008, pp. 448–453. AAAI Press (2008)
6. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**(3/4), 365–386 (1991)
7. Oetsch, J., Pührer, J., Tompits, H.: Stepwise debugging of answer-set programs. Theory Pract. Log. Program. **18**(1), 30–80 (2018)
8. Shchekotykhin, K.M.: Interactive query-based debugging of ASP programs. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 25–30 January 2015, Austin, Texas, USA, pp. 1597–1603. AAAI Press (2015)
9. Thevapalan, A., Heyninck, J., Kern-Isberner, G.: Establish coherence in logic programs modelling expert knowledge via argumentation. In: Workshop on Causal Reasoning and Explanation in Logic Programming (CAUSAL 2021) (2021)
10. Thevapalan, A., Kern-Isberner, G.: Towards interactive conflict resolution in ASP programs. In: Martínez, M.V., Varzinczak, I. (eds.) Proceedings of the 18th International Workshop on Non-Monotonic Reasoning, NMR 2020, pp. 29–36, September 2020