On Establishing Robust Consistency in Answer Set Programs

ANDRE THEVAPALAN and GABRIELE KERN-ISBERNER

Technische Universität Dortmund, Dortmund, Germany

(*e-mails:* andre.thevapalan@tu-dortmund.de, gabriele.kern-isberner@cs.uni-dortmund.de)

Abstract

Answer set programs used in real-world applications often require that the program is usable with different input data. This, however, can often lead to contradictory statements and consequently to an inconsistent program. Causes for potential contradictions in a program are conflicting rules. In this paper, we show how to ensure that a program $\mathcal{P}$ remains non-contradictory given any allowed set of such input data. For that, we introduce the notion of conflict-resolving λ-extensions. A conflict-resolving λ-extension for a conflicting rule r is a set λ of (default) literals such that extending the body of r by λ resolves all conflicts of r at once. We investigate the properties that suitable λ-extensions should possess and building on that, we develop a strategy to compute all such conflict-resolving λ-extensions for each conflicting rule in $\mathcal{P}$. We show that by implementing a conflict resolution process that successively resolves conflicts using λ-extensions eventually yields a program that remains non-contradictory given any allowed set of input data.

KEYWORDS: logic programming, answer set programming, consistency, contradictions, conflicts, interactions

1 Introduction

1.1 Motivation and context

Answer set programs can be used to implement real-world applications like decision support systems that aid knowledge experts whenever crucial decisions based on different rules and conditions have to be made. However, the knowledge bases of such applications are not static but rather very dynamic in the sense that they are adapted to each individual case and can also be prone to various updates. Especially in domains like the medical sector, knowledge bases are expected to yield suitable decisions for each patient where patients can show diverse symptoms (implemented by facts), and the knowledge base is to be updated very often, for example, every month. In addition, physicians are usually faced with some degree of uncertainty and for example have to rely on own experience when they have to reach important decisions (e.g. regarding the treatment of patients) (Ghosh 2004). Imagine a system that outputs the possible treatment plans for a given patient based on a corresponding medical ruleset. For each patient,

the application has to combine the general knowledge about possible treatment plans (*problem encoding*) with the data regarding the patient (*problem instance*) (Gebser *et al.* 2012). Furthermore, the knowledge base containing the encoding can grow over time, not only adding new knowledge but also revising or removing deprecated knowledge. Additionally, decision-making processes in this sector do not only consist of, for example, finding the right therapy for each patient, it is also important to assure that unfitting solutions are also reflected as such (e.g. what therapies must not be recommended to a patient). This emphasizes how the usage of an answer set program with both default and strong negation constitutes a highly valuable asset in assisting medical experts.

However, at any point where such an application is used, it has to be ensured that the respective knowledge base remains consistent when merged with the patient data. For a knowledge base that is frequently updated and that also has to be used with different instance data, maintaining consistency can become a quite cumbersome task as it requires not only a complete understanding of the whole problem encoding but also technical knowledge regarding logic programming. In previous approaches regarding the update of answer set programs (Eiter *et al.* 2002; Alferes *et al.* 1998), every update required an automated adaptation of the updated program in order to prevent conflicts. Using these methods for a medical support system would mean that the changes that are made in the new program are not managed and approved by the knowledge expert. Naturally, these inconveniences can deter knowledge experts from using answer set programs for the implementation even though answer set programming itself is well suited for highly complex decision-making problems.

It is for this reason that we propose a general framework that allows the maintenance of a logic program by involving the expertise of knowledge experts more directly. Such maintenance tasks would include the detection of possible conflicting statements between the problem encoding and any possible instance data and the resolution of such conflicts. Note that the consistency of an answer set program crucially depends on its facts, in particular, its input data. Conflicts between rules may become apparent only if specific input data is provided, resulting in an unexpected failure in a specific (rare) case of an otherwise helpful and approved program. Our approach aims at anticipating such conflicts, ensuring that a program yields professionally adequate solutions for any (future) case. The resolution of such conflicts should be overseen by the knowledge expert and executed in an interactive fashion. To facilitate the conflict resolution, possible solutions for each conflict should be generated and presented to the expert on demand. They can then choose a most suitable solution. In this way it is guaranteed that every modification that is applied to the knowledge base is valid in a professional sense. Such an interactive exchange between the system and the expert during the resolution process can therefore eliminate the need for a deeper technical understanding of logic programming, giving the knowledge expert full control over the maintenance operations, guaranteeing that no changes are done blindly. Especially in fields like the medical sector, full expert control over the knowledge base is crucial as small mistakes can have serious consequences. The gap between a professional and a technical expert can be significant and make the automatic resolution of conflicts by suitable modifications of the program impossible. So, extensive communications between the two experts would be needed. The proposed framework constitutes a first step towards closing this gap as the conflict resolution process can be fully executed by the professional expert directly

because all proposed modifications will be technically viable solutions of the conflict, and the expert can then choose from them according to professional standards. Being able to establish the consistency of the problem encoding for every possible instance data without actually requiring the explicit instance has the effect that, in practice, the program does not have to be validated for each instance, which makes it much more robust and reliable.

1.2 Main contributions

In order to facilitate the integration of answer set programming in real-world applications, in this paper, we show how to ensure that the problem encoding in a logic program remains consistent given any (allowed) set of consistent instance data. Inspired by Eiter *et al.* (2002) and Alferes *et al.* (1998, 2000) and their work on logic program updates, we develop a strategy to extend rules of a logic program such that the derivation of contradictory statements is no longer possible in any admissible case. The presented approach expands previous approaches in two major ways:

(1) Programs that are modified following our approach remain consistent for any given input data that do not contain atoms that appear in head literals of the program. For that, we extend the notion of literals and investigate the relationship between the different negated versions of a literal. This enables us to define the characteristics of conflicting and non-conficting rules, and in particular to determine how the body of a rule r_i that is in conflict with several rules r_j can be extended such that the modified rule r_i' is no longer conflicting with any rule r_j while the extension comprises only informative literals. By informative, we mean that these extensions should only be composed of literals whose atoms appear in the body of the rules involved in the conflict. For that reason, we also adapt the notion of *hitting sets* for (default) literals in logic program rules. As a result, the meaningfulness of the problem encoding is validated beforehand, eliminating the need for testing the encoding against all possible instances or other precautionary measures for each instance.

(2) Instead of using a technical device to prioritize one statement over another in case of a conflict like *causal rejection (Eiter et al. 2002)*, we show how all *informative* extensions for a conflicting rule can be computed. As a consequence, a knowledge expert could then be included into the resolution process and choose the suggested rule modification that is most suitable such that the resulting modified program remains professionally adequate. Using informative extensions also maintains the readability of the rules in a program which in turn can simplify subsequent update operations.

(3) The final goal is the construction of a framework for the interactive maintenance of large ASP knowledge bases which can be used by knowledge experts without the need for technical knowledge and where each modification can be overseen by the expert. The approach presented in this paper is a basic building block for such a framework as the computation of all possible informative extensions can be used to resolve each conflict in cooperation with the knowledge expert which in turn guarantees that the resulting knowledge base still contains professionally adequate knowledge. We will also show that the approach follows a pragmatic paradigm which allows various extensions in order to facilitate the conflict resolution process for the expert.

1.3 Structure of the paper

The paper is organized as follows: We begin by providing the necessary preliminaries about extended logic programs in Section 2. In Section 3, we present our conflict resolution approach. We start in Section 3.1 by introducing the notion of uniformly non-contradictory program cores and examine the properties of conflicts and non-conflicting rules in Section 3.2 before we connect these results in Section 3.3 by defining the properties of a conflict resolution step. After describing a naïve conflict resolution approach in Section 3.4 by using semi-normal completions (Caminada 2006), we present in Section 3.5 our strategy to compute all appropriate rule modifications of conflicting rules in order to obtain a uniformly non-contradictory program core. In Section 4, we present additional ways to use and extend the conflict resolution approach for the usage in real-world applications. Sections 4.1 and 4.2 show how the presented method can be used to resolve many-to-many conflicts and how inconsistency that can be caused by a specific type of constraints can be prevented. Sections 4.3 and 4.4 outline ways to enhance the handling of λ-extensions in order to facilitate their usage in applications. The paper concludes with a summary and a discussion regarding future work.

2 Preliminaries

2.1 Extended Logic Programs

In this paper, we look at non-disjunctive *extended logic programs* (ELPs) (Gelfond and Lifschitz 1991). An ELP is a finite set of rules over a set $\mathcal{A}$ of propositional atoms. First, we discuss the different forms of negation in ELPs and introduce notations. 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. For a set S of classical literals, $\overline{S} = \{\overline{L} \mid L \in S\}$ is the set of corresponding strongly complementary literals. Then, $Lit_{\mathcal{A}}$ denotes the set $\mathcal{A} \cup \overline{\mathcal{A}}$ of all classical literals over $\mathcal{A}$. A *default-negated literal* L, called *default literal*, is written as $\sim L$. In logic programs, $\sim$ will be used as a prefix solely for classical literals, symbolizing the default negation which is usually denoted by the prefix *not* (Gelfond and Lifschitz 1991). Outside of logic programs, we will use $\sim$ as a unary junctor in order to describe the *default complement* of a (default) literal $(\sim)L$, that is, the default complement of L is $\sim L$ and the default complement of $\sim L$ is $\sim\sim L = L$. This reflects the binary characteristic of the default negation which is illustrated in Figure 1. For a set S of classical literals, we define $\sim$ accordingly, that is, $\sim S = \{\sim L \mid L \in S\}$. Given a set S of (classical) literals, we say a (classical) literal L *is true in* S (symbolically $S \vDash L$) iff $L \in S$ and $\sim L$ is true in S (symbolically $S \vDash \sim L$) iff $L \notin S$. By an *extended literal* L^*, we either mean a (classical) literal L or a default-negated literal $\sim L$. The set of all extended literals over a set of atoms $\mathcal{A}$ will be denoted by $Lit_{\mathcal{A}}^*$, that is, $Lit_{\mathcal{A}}^* = Lit_{\mathcal{A}} \cup \sim Lit_{\mathcal{A}}$. A set X of extended literals is true in S (symbolically $S \vDash X$) iff every extended literal $L^* \in X$ is true in S. With $atom(L^*)$, we will associate the atom on which the extended literal L^* is based on. The underlying atoms of a set X of extended literals is given by the set of atoms $Atom(X) = \{atom(L^*) \mid L^* \in X\}$. Two extended literals L^*, K^* are *atom-related* if $atom(L^*) = atom(K^*)$.

Fig. 1. Visualization of truth values of an atom A w.r.t. a set of literals S.
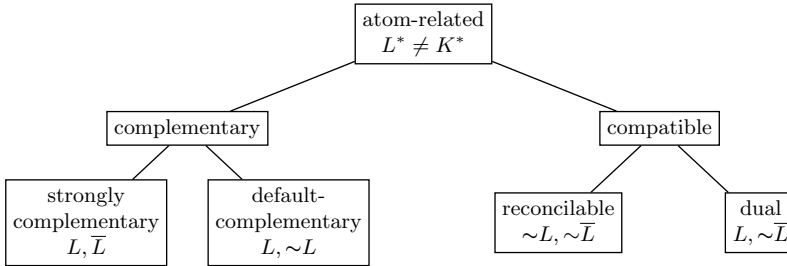

Fig. 2. Possible reationships between two atom-related literals.

The following definition introduces handy terms to describe the relationships between literals.

Definition 1
Given a classical literal L, we say that

- L and $\sim L$ are *default complementary*,
- L and $\sim\overline{L}$ are *dual*, and
- $\sim L$ and $\sim\overline{L}$ are *reconcilable*.

Given two atom-related literals L^*, K^* with $L^* \neq K^*$, we will say L^* and K^* are

- *complementary* if L^*, K^* are strongly or default complementary, and
- *compatible* if L^*, K^* are reconcilable or dual.

A set of classical literals is *inconsistent* if it contains strongly complementary literals. A set of extended literals is *inconsistent* if it contains complementary literals.

Figure 2 visualizes the different relationships that two atom-related literals L^*, K^* can have. The different negation types and the particular significance of reconcilable literals will be discussed in Section 3.3.

We are now ready to specify the form of ELPs.

A *rule* r is of the form

$$L_0 \leftarrow L_1, \dots, L_m, \sim L_{m+1}, \dots, \sim L_n., \tag{1}$$

with classical literals $L_0, \dots, L_n$ and $0 \leq m \leq n$. The literal L_0 is the *head* of r, denoted by $H(r)$, and $\{L_1, \dots L_m, \sim L_{m+1}, \dots \sim L_n\}$ is the *body* of r, denoted by $B(r)$. Furthermore, $\{L_1, \dots, L_m\}$ is denoted by $B^+(r)$ and $\{L_{m+1}, \dots, L_n\}$ by $B^-(r)$. Given a set of rules $\mathcal{R} \subseteq \mathcal{P}$, we will denote the set of all extended literals occurring in the rule bodies of $\mathcal{R}$ by $B^*(\mathcal{R})$, that is, $B^*(\mathcal{R}) = \bigcup_{r \in \mathcal{R}} B(r)$. A rule r with $B(r) = \emptyset$ is called a *fact*, and r is called a *constraint* if it has an empty head. For a fact $L.$, we will call the corresponding

literal L a *fact literal*. A set $\mathcal{F}$ of facts is *consistent* if the set of fact literals in $\mathcal{F}$ is consistent. A rule r will be called a *complex rule* if r is neither a fact nor a constraint. An *extended logic program (ELP)* is a set of rules of the form (1).

The program $\mathcal{P}^+$ will denote the reduction of $\mathcal{P}$ to a *normal program* (a logic program without classically negated literals) that is obtained by replacing every classically negated literal $\overline{A}$ in $\mathcal{P}$ by a new corresponding atom A' (Gelfond and Lifschitz 1991).

A rule r is *applicable* iff there is a consistent set S of classical literals such that $S \vDash B(r)$. For the rest of this paper, we will assume that every rule in a given logic program is applicable. Given a set S of classical literals, a (complex) rule r *is true in S* (symbolically $S \vDash r$) iff $H(r)$ is true in S whenever $B(r)$ is true in S. In case r is a constraint, r is true in S iff $S \nvDash B(r)$. Whenever a rule r is true in S we also say that S *satisfies r*. A rule body $B(r)$ will be called *satisfiable* if there exists a consistent set S of classical literals such that $S \vDash B(r)$. Correspondingly, given a set of rules $\mathcal{R} = \{r_1, \ldots, r_n\}$, the bodies of all rules in $\mathcal{R}$ are *simultaneously satisfiable* whenever there is a set S of classical literals such that $S \vDash B(r)$ for every rule $r \in \mathcal{R}$. Given an ELP $\mathcal{P}$ without default negation, the *answer set* of $\mathcal{P}$ is either (a) the smallest set $S \subseteq Lit_{\mathcal{A}}$ such that S is consistent and $S \vDash r$ for every rule $r \in \mathcal{P}$, or (b) the set $Lit_{\mathcal{A}}$ of classical literals. Note that similar to Horn logic programs, each such ELP has exactly one minimal model which might, however, be inconsistent.

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 classical literals is defined by

$$\mathcal{P}^S = \{H(r) \leftarrow B^+(r). \mid r \in \mathcal{P}, B^-(r) \cap S = \emptyset\}. \tag{2}$$

A set S of classical literals is an *answer set* of $\mathcal{P}$ if it is the answer set of $\mathcal{P}^S$ (Gelfond and Lifschitz 1991). The set of all answer sets of a program $\mathcal{P}$ will be denoted by $AS(\mathcal{P})$. Note that since every inconsistent set of literals that satisfies all rules in $\mathcal{P}$ will be replaced by $Lit_{\mathcal{A}}$, every reduct of $\mathcal{P}$ is formed relative to either a consistent set S or $Lit_{\mathcal{A}}$. As a consequence, $AS(\mathcal{P})$ can either consist of only consistent sets of literals, only $Lit_{\mathcal{A}}$, or no sets at all. We say a classical literal L is *derivable* in $\mathcal{P}$ iff $L \in \bigcup AS(\mathcal{P})$. For a set T of classical literals, we say all literals in T are *simultaneously derivable* iff there exists an answer set $S \in AS(\mathcal{P})$ such that $T \subseteq S$. Atoms that occur in the rule bodies of $\mathcal{P}$ but not in any rule head will be called *external atoms* and the set of external atoms will be denoted by $\mathcal{E}_{\mathcal{P}}$. Atoms that occur in $\mathcal{P}$ that are not external will be called *internal atoms*. As stated in van Gelder *et al.* (1991), deductive databases are commonly viewed as logic programs as they consist of an *external database*, which is a set of facts, and the *internal database*, which is a set of rules. In software applications, often the same program is used with different input data. This can also be applied to extended logic programs. Similar to the division inside databases and the distinction between *problem instance and encoding* as mentioned before, we partition an ELP $\mathcal{P}$ into a set $\mathcal{P}_{\mathcal{F}}$ of facts which we will call *input*, and a set $\mathcal{P}_{\mathcal{C}}$ of rules which will be called *program core*. We define that any *valid input* $\mathcal{P}_{\mathcal{F}}$ for $\mathcal{P}_{\mathcal{C}}$ is a consistent set of facts over $Lit\mathcal{E}_{\mathcal{P}}$ and that $\mathcal{P}_{\mathcal{C}}$ can only comprise complex rules. The set of all valid inputs $\mathcal{P}_{\mathcal{F}}$ for a program core $\mathcal{P}_{\mathcal{C}}$ will be denoted by $I(\mathcal{P}_{\mathcal{C}})$. By $\Pi(\mathcal{P}_{\mathcal{C}}) = \{\mathcal{P}_{\mathcal{C}} \cup \mathcal{P}_{\mathcal{F}} \mid \mathcal{P}_{\mathcal{F}} \in I(\mathcal{P}_{\mathcal{C}})\}$, we denote the set of all programs $\mathcal{P}_{\mathcal{C}}$ extended by a valid input $\mathcal{P}_{\mathcal{F}}$ for $\mathcal{P}_{\mathcal{C}}$.

In Section 3.1, we will explain why it is necessary to restrict $\mathcal{P}_{\mathcal{F}}$ to literals in $Lit\mathcal{E}_{\mathcal{P}}$.

As mentioned above, default negation puts a classical literal L into a binary state w.r.t. a set S of classical literals. Either L is true in S while $\sim L$ is false in S or vice versa. With regard to strong negation however, this is not the case. In Figure 1, this distinction is illustrated by the *gap* between *true* and *false* on the level of A and $\overline{A}$. Under three-valued semantics (Przymusinski 1991), this additional *state* is referred to as *undefined*. Atoms A and $\overline{A}$ are undefined in S if neither A nor $\overline{A}$ are derivable in S. Under answer set semantics, this undefinedness is covered by default literals, that is, both $\sim A$ and $\sim \overline{A}$ are simultaneously true whenever neither A nor $\overline{A}$ is derivable. It is for this reason that we introduced the notion of reconcilable literals. Even though for two reconcilable literals $\sim A, \sim \overline{A}$, the atoms $A, \overline{A}$ "inside" the literals are strongly complementary, the default negation allows that both literals are true simultaneously in S, that is, $S \vDash \{\sim A, \sim \overline{A}\}$ whenever $A, \overline{A} \notin S$.

2.2 Consistency

In Gelfond and Lifschitz (1991), an answer set program $\mathcal{P}$ over $\mathcal{A}$ is *inconsistent* if $\mathcal{P}$ has no answer sets or its only answer set is $Lit_{\mathcal{A}}$.

We will specify inconsistency in more detail by using the results in Schulz *et al.* (2015) where the authors define four different cases how and why an ELP can be inconsistent. There, in the first three cases *1, 2* and *3a*, the inconsistency is essentially caused by classically complementary literals in the program, that is, by their simultaneous derivation. In the fourth case *3b*, a so called *negative dependency path* exists that leads to the derivation of default-complementary literals. From a semantic viewpoint, in case *1* and *2*, $\mathcal{P}$ has no well-founded models. In case *3a*, $\mathcal{P}$ has a well-founded model, and the normal program $\mathcal{P}^+$ of $\mathcal{P}$ has one or more answer sets. In case *3b*, $\mathcal{P}$ has a well-founded model but $\mathcal{P}^+$ has no answer sets. We will say that an inconsistent ELP $\mathcal{P}$ is *contradictory* if the inconsistency is caused by classically complementary literals (case 1, 2 or 3a). Otherwise, $\mathcal{P}$ will be called *incoherent* (case 3b).[1] In this work, we will only consider contradictory programs. The handling of incoherent programs is examined in Thevapalan *et al.* (2021) as well as in Costantini (2006).

The presented method will guarantee that a modified ELP $\mathcal{P}$ is not contradictory with any valid input $\mathcal{P}_{\mathcal{F}}$. For this reason, we introduce the notion of *uniformly non-contradictory program cores*.

Definition 2 (Uniformly Non-Contradictory Program Core)
A program core $\mathcal{P}_{\mathcal{C}}$ of an ELP $\mathcal{P}$ over $\mathcal{A}$ is *uniformly non-contradictory* if for every valid input $\mathcal{P}_{\mathcal{F}}$ for $\mathcal{P}_{\mathcal{C}}$, $\mathcal{P}_{\mathcal{C}} \cup \mathcal{P}_{\mathcal{F}}$ is not contradictory.

Corollary 1
Given a uniformly non-contradictory program core $\mathcal{P}_{\mathcal{C}}$ over $\mathcal{A}$, for every $\mathcal{P} \in \Pi(\mathcal{P}_{\mathcal{C}})$ the following holds: either $S \neq Lit_{\mathcal{A}}$ for every $S \in AS(\mathcal{P})$, or $\mathcal{P}$ is incoherent.

Naturally, given a program core $\mathcal{P}_{\mathcal{C}}$, every program $\mathcal{P} \in \Pi(\mathcal{P}_{\mathcal{C}})$ is uniformly non-contradictory whenever $\mathcal{P}_{\mathcal{C}}$ is.

---

[1] Note that the inconsistency type is not solely determined by the answer set semantics of $\mathcal{P}$ but also by using well-founded semantics and the normal program of $\mathcal{P}$. For more information on the inconsistency types, we refer the reader to Schulz *et al.* (2015), Inoue (1993).

As stated in Gelfond and Lifschitz (1991) and implied by Corollary 1, non-contradictory program cores do not guarantee consistent programs since an ELP that is free of contradictions could potentially still be incoherent. Detecting incoherence and establishing coherence in an ELP, though, is a separate task (see Schulz *et al.* 2015; Schulz 2017) and outside of the scope of this paper. We, therefore, assume that in the following, any given ELP is coherent if not stated otherwise.

3 Contradictions

The aim of this paper is to present a method that analyzes program cores and computes possible modifications such that the modified program core becomes uniformly non-contradictory. The presented method will, therefore, ensure that given a program core $\mathcal{P}_\mathcal{C}$, there is no contradictory ELP $\mathcal{P} \in \Pi(\mathcal{P}_\mathcal{C})$. To that end, we first *identify* what may cause contradictions. We will then show how to find the causes (*conflict detection*) and remove them (*conflict resolution*). In this section, we will assume that any given program core $\mathcal{P}_\mathcal{C}$ does not contain constraints. The influence of constraints with respect to the consistency of the program will be examined in Section 4.2.

3.1 Conflicts

To find rules that could potentially lead to contradictions, one has to look at rules with strongly complementary head literals. We will call two rules with strongly complementary head literals *conflicting* if both rule bodies are simultaneously satisfiable by a consistent set of classical literals.

Definition 3 (*Conflicting Rules, Conflict Thevapalan and Kern-Isberner 2020*)
Suppose an ELP $\mathcal{P}$. Two rules $r, r' \in \mathcal{P}_\mathcal{C}$, $r \neq r'$, are *conflicting* (written as $r \bowtie r'$) if $H(r)$ and $H(r')$ are strongly complementary and there exists a consistent set of classical literals $S \subseteq Lit_\mathcal{A}$ such that $B(r)$ and $B(r')$ are true in S. A *conflict* is a pair (r, r') of rules such that r, r' are conflicting. We will denote the set of all conflicts (r, r') in an ELP $\mathcal{P}$ by $Conflicts(\mathcal{P})$, and correspondingly the set of all conflicts (r, r') involving a rule r will be denoted by $conflicts(r)$. Furthermore, we will refer to the set of rules $r' \in \mathcal{P}$ that are conflicting with r in $\mathcal{P}$ as *adversarial rules* of r, denoted by $Adv(r)$, that is, $Adv(r) = \{r' \mid r \bowtie r', r' \in \mathcal{P}\}$. We will call two rules r, r' *non-conflicting* (symbolically $r \not\bowtie r'$) iff $r \bowtie r'$ does not hold.

Remark 1: (a) If $r, r' \subseteq \mathcal{P}_\mathcal{C}$ are conflicting rules, then there exists a consistent set S of classical literals such that $S \vDash B(r), B(r')$. Consequently, there is a set of facts $\mathcal{F}$ (obtained from S in a straighforward way) such that $\mathcal{P}_\mathcal{C} \cup \mathcal{F}$ is contradictory. (b) The identification of two rules as conflicting is done independently of the rest of the program core and especially independently of any set $\mathcal{F}$ of currently given facts.

Proposition 1
Given an ELP $\mathcal{P}$ over $\mathcal{A}$, its program core $\mathcal{P}_\mathcal{C}$ is uniformly non-contradictory if $\mathcal{P}_\mathcal{C}$ does not contain any conflicts.

Table 1. *Relationship between two rules r, r' with complementary head literals w.r.t. a literal $L = A$*

	$L \in B^+(r)$	$\overline{L} \in B^+(r)$	$L \in B^-(r)$	$\overline{L} \in B^-(r)$
$L \in B^+(r')$	$r \bowtie r'$ (possibly)	$r \not\bowtie r'$	$r \not\bowtie r'$	$r \bowtie r'$ (possibly)
$\overline{L} \in B^+(r')$	$r \not\bowtie r'$	$r \bowtie r'$ (possibly)	$r \bowtie r'$ (possibly)	$r \not\bowtie r'$
$L \in B^-(r')$	$r \not\bowtie r'$	$r \bowtie r'$ (possibly)	$r \bowtie r'$ (possibly)	$r \bowtie r'$ (possibly)
$\overline{L} \in B^-(r')$	$r \bowtie r'$ (possibly)	$r \not\bowtie r'$	$r \bowtie r'$ (possibly)	$r \bowtie r'$ (possibly)

Proof

Assume an ELP $\mathcal{P}$ whose core $\mathcal{P}_{\mathcal{C}}$ is not uniformly non-contradictory. Then, by Definition 2, there exists a valid input $\mathcal{P}_{\mathcal{F}}$ such that $\mathcal{P} = \mathcal{P}_{\mathcal{C}} \cup \mathcal{P}_{\mathcal{F}}$ is contradictory. This implies that there exist two complementary literals $L, \overline{L}$ that are simultaneously derivable in $\mathcal{P}$. $L, \overline{L}$ are only simultaneously derivable if either (a) $\{L., \overline{L}.\} \subseteq \mathcal{P}_{\mathcal{F}}$, or (b) $L. \in \mathcal{P}_{\mathcal{F}}$ and there exists a rule $r \in \mathcal{P}_{\mathcal{C}}$ with $H(r) = \overline{L}$ such that $B(r)$ is satisfied in $\mathcal{P}$, or (c) there exist two rules $r, r' \in \mathcal{P}_{\mathcal{C}}$ with $H(r) = L, H(r') = \overline{L}$ such that $B(r), B(r')$ are satisfied in $\mathcal{P}$. Regarding (a): $\mathcal{P}_{\mathcal{F}}$ cannot contain fact literals $L, \overline{L}$ because $\mathcal{P}_{\mathcal{F}}$ has to be consistent by definition. Regarding (b): $\mathcal{P}_{\mathcal{F}}$ does only contain facts over external atoms. Since, by definition, external atoms do not occur in rule heads of $\mathcal{P}_{\mathcal{C}}$, $L.$ cannot be in $\mathcal{P}_{\mathcal{F}}$ whenever there is a rule in $\mathcal{P}_{\mathcal{C}}$ with head literal $\overline{L}$. Therefore, the only way how $L, \overline{L}$ could be derived simultaneously is if (c) holds, that is, if there exist two rules r, r' in $\mathcal{P}_{\mathcal{C}}$ with complementary rule heads that are satisfiable in $\mathcal{P}$. Consequently, if $\mathcal{P}_{\mathcal{C}}$ is not uniformly non-contradictory, then $\mathcal{P}_{\mathcal{C}}$ contains at least one conflict. $\qquad\square$

In nonmonotonic logics, the possible occurence of contradictions is often dealt with by restricting the syntax of knowledge representation languages. As mentioned before, restricting the set of possible facts in $\mathcal{P}_{\mathcal{F}}$ to facts over external atoms of the respective program core limits the expressibility in programs. But this limitation assures that contradictions via a rule in $\mathcal{P}_{\mathcal{C}}$ and a fact in $\mathcal{P}_{\mathcal{F}}$ cannot arise even if $\mathcal{P}_{\mathcal{C}}$ is conflict-free since those kinds of contradictions are not caused by conflicts between complex rules in $\mathcal{P}_{\mathcal{C}}$. Therefore, in the following, $\mathcal{P}_{\mathcal{F}}$ will only consist of fact literals over $\mathcal{E}_{\mathcal{P}}$.

3.2 Conflict detection

We will now analyze the properties of (non-)conflicting rules. For two rules with complementary heads to be conflicting, both their bodies have to be satisfiable by at least one consistent set of literals. Table 1 shows the different combinations in which an atom $A \in \mathcal{A}$ can appear as a classical literal L in an ELP $\mathcal{P}$ over $\mathcal{A}$ in the bodies of two rules r, r' with complementary rule heads. For each case, the table states whether both rule bodies can hold simultaneously and consequently, whether L makes the two rules explicitly non-conflicting, or a conflict might be possible. It is easy to see that r, r' are non-conflicting whenever the corresponding extended literals in $B(r), B(r')$ that are based on A are strongly or default complementary. These observations lead to the following conclusion:

Theorem 1
Let $\mathcal{P}$ be a program with rules $r, r' \in \mathcal{P}_\mathcal{C}$. Two rules r, r' are conflicting if and only if

(CP1) $H(r), H(r')$ are strongly complementary, and
(CP2) $B^+(r_1) \cap B^-(r_2) = \emptyset$ such that $r_1, r_2 \in \{r, r'\}, r_1 \neq r_2$, and
(CP3) $B^+(r) \cup B^+(r')$ is consistent.

Proof
Suppose an ELP $\mathcal{P}$ over $\mathcal{A}$, and rules r, r' in $\mathcal{P}_\mathcal{C}$ that satisfy (CP1), (CP2), and (CP3). For rules r, r' to be conflicting according to Definition 3, their head literals have to be strongly complementary, that is, $L = H(r)$ and $\overline{L} = H(r')$. This is satisfied via (CP1). Definition 3 also requires that for r, r' to be conflicting, there has to exist a consistent set $S \subseteq Lit_\mathcal{A}$ of classical literals for which the bodies $B(r), B(r')$ are true. We show that $S = B^+(r) \cup B^+(r')$ is such a set. By (CP3), S is consistent. In order for S to satisfy both rule bodies $B(r), B(r')$ simultaneously, S has to have the following properties by definition: (a) $S \cap (B^-(r) \cup B^-(r')) = \emptyset$ and (b) $B^+(r) \cup B^+(r') \subseteq S$. (b) is trivially fulfilled. Regarding (a), we determine

$$
\begin{aligned}
S \cap (B^-(r) \cup B^-(r')) &= (B^+(r) \cup B^+(r')) \cap (B^-(r) \cup B^-(r')) \\
&= (B^+(r) \cap B^-(r)) \cup (B^+(r) \cap B^-(r')) \\
&\quad \cup (B^+(r') \cap B^-(r)) \cup (B^+(r') \cap B^-(r')).
\end{aligned}
\tag{3}
$$

Due to our assumption (see Remark 1), the first and the last intersection in (3) are empty, and due to (CP2), also the middle ones in (3) are empty. Altogether, we have $S \cap (B^-(r) \cup B^-(r')) = \emptyset$ which yields (a). So, (r, r') is a conflict in $\mathcal{P}$.

Conversely, let $\mathcal{P}$ be an ELP with conflicting rules $r, r' \in \mathcal{P}_\mathcal{C}$. (CP1) holds, since by definition, the head literals of r and r' are strongly complementary. Furthermore, there exists a consistent set S of classical literals such that $B(r)$ and $B(r')$ are true in S. Since S cannot satisfy two classical literals L and $\sim L$ simultaneously, (CP2) holds. As S is consistent, its subset $B^+(r) \cup B^+(r')$ is also consistent. Therefore, (CP3) holds. $\qquad \square$

Given (CP2) and (CP3) in Theorem 1 and Table 1, we can see that r, r' are not conflicting whenever $B(r)$ contains a literal L such that $B(r')$ contains a literal that is either strongly or default-complementary to L.

Definition 4 (Conflict-Preventing Literals)
Given an ELP $\mathcal{P}$ and two rules $r, r' \in \mathcal{P}_\mathcal{C}$ with complementary heads, two extended literals $L^* \in B(r), K^* \in B(r')$ are *conflict-preventing* if L^* and K^* are atom-related and complementary.

This leads us to the following observation:

Proposition 2
Let $\mathcal{P}$ be an ELP with two rules $r, r' \in \mathcal{P}$ with complementary heads. The rules r, r' are non-conflicting iff there exist two extended literals $L^* \in B(r), K^* \in B(r)$ such that L^*, K^* are conflict-preventing.

Proof
Two conflict-preventing literals $L^* \in B(r)$, $K^* \in B(r')$ are either (a) default-complementary or (b) strongly complementary. In case of (a), the rules r, r' will not satisfy

(CP2) in Definition 3, and in case of (b), (CP3) is not met. On the other hand, if two rules r, r' with complementary head literals are non-conflicting, then by Definition 3 either condition (CP2) or (CP3) or both are not met. As shown before, (CP2) (resp. (CP3)) can only be not satisfied by r, r' if their bodies contain default-complementary (resp. strongly complementary) literals L^*, K^*. In both cases, L^*, K^* are conflict-preventing literals. □

Example 1
Consider the ELP $\mathcal{P}_1$:

$$r_1 \colon x \leftarrow a, {\sim}b.$$
$$r_2 \colon \overline{x} \leftarrow b, {\sim}c.$$
$$r_3 \colon \overline{x} \leftarrow \overline{a}, {\sim}b.$$
$$r_4 \colon \overline{x} \leftarrow a, \overline{b}.$$
$$r_5 \colon \overline{x} \leftarrow c, d.$$

$\mathcal{P}_1$ has two conflicts: (r_1, r_4) and (r_1, r_5). The rules r_1, r_2 violate (CP2) and have the conflict-preventing literals $b, {\sim}b$. The rules r_1, r_3 violate (CP3) and have the conflict-preventing literals $a, \overline{a}$.

3.3 Conflict resolution

Next, we will show how to resolve conflicts in an ELP $\mathcal{P}$. To resolve all conflicts in $\mathcal{P}$, one or both rules of each conflict in $Conflicts(\mathcal{P})$ have to be modified such that $\mathcal{P}_{\mathcal{C}}$ becomes uniformly non-contradictory. We will call the modification of $\mathcal{P}$ to $\mathcal{P}'$ a *conflict resolution step* whenever at least one conflict is resolved and a sequence of conflict resolution steps $\langle \mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \dots, \mathcal{P}^{(n)} \rangle$ will be called *conflict resolution process*. We present an approach to compute possible solutions to resolve a conflict in $\mathcal{P}$ such that the following properties hold:

(P1) The conflict resolution process is *successful*. By that, we mean that a conflict resolution process where in each step a computed resolution option is applied will eventually lead to a non-contradictory program core, that is, a finite sequence of conflict resolution steps $\langle \mathcal{P}^{(1)}, \dots, \mathcal{P}^{(n)} \rangle$ such that $Conflicts(\mathcal{P}^{(n)}) = \emptyset$.

(P2) Each conflict resolution step is *minimally invasive* as it only consists of extending the body of a conflicting rule. This means, for every rule $r \in \mathcal{P}_{\mathcal{C}}$ with $conflicts(r) \neq \emptyset$, the corresponding modified rule $r' \in \mathcal{P}'_{\mathcal{C}}$ satisfies that $H(r') = H(r)$ and $B(r') \supseteq B(r)$, and for every other rule r, it holds that $r' = r$.

(P3) Each rule that is modified during the resolution process remains *applicable*.

Given an ELP $\mathcal{P}$ over a set of atoms $\mathcal{A}$ and a rule $r_i \in \mathcal{P}_{\mathcal{C}}$ with a non-empty set $conflicts(r_i)$, we want to modify r_i to a rule r'_i such that every conflict in $conflicts(r_i)$ is resolved, that is, $conflicts(r'_i) = \emptyset$. For that, our approach analyzes how the extended literals $Lit^*_{\mathcal{A}}$ of $\mathcal{P}$ jointly appear in the rules of $Adv(r_i)$, and computes *conflict-resolving extensions* for r_i.

Definition 5
Given an ELP $\mathcal{P}$ over $\mathcal{A}$ and a rule $r_i \in \mathcal{P}$, a *λ-extension $\lambda(r_i)$ for r_i* is a set of extended literals $L^* \in Lit\mathcal{E}_{\mathcal{P}}$ such that $Atom(\lambda(r_i)) \cap Atom(B(r_i)) = \emptyset$. A rule r'_i of the form

$$r_i': H(r_i) \leftarrow B(r_i), \lambda(r_i). \tag{4}$$

will be called a λ-*extended rule w.r.t.* $\lambda(r_i)$.

To resolve all conflicts $(r_i, r_j) \in conflicts(r_i)$ of a rule $r_i \in \mathcal{P}$, we want to gather those λ-extensions $\lambda(r_i)$ for r_i such that the λ-extended rule r_i' w.r.t. $\lambda(r_i)$ and each r_j are not conflicting, that is, $conflicts(r_i') = \emptyset$.

Definition 6
Given an ELP $\mathcal{P}$ over $\mathcal{A}$ and a rule $r_i \in \mathcal{P}$ with $Adv(r_i) \neq \emptyset$, a λ-extension $\lambda(r_i)$ for r_i is *conflict-resolving* iff $Adv(r_i') = \emptyset$ where r_i' is the λ-extended rule w.r.t. $\lambda(r_i)$. We say a conflict-resolving λ-extension for r_i *resolves all conflicts in* $conflicts(r_i)$ *simultaneously*.

Hence, in order to obtain a uniformly non-contradictory program core $\mathcal{P}_\mathcal{C}$, we resolve all conflicts in $\mathcal{P}_\mathcal{C}$ by extending the bodies of particular conflicting rules by a respective conflict-resolving λ-extension.

We can show that extending a rule in $\mathcal{P}$ by a λ-extension does not lead to additional conflicts.

Proposition 3
Let $\mathcal{P}_\tau = \mathcal{P} \backslash \{r_i\}$ be the set of all rules in $\mathcal{P}$ other than r_i. Let furthermore r_i' be the λ-extended rule w.r.t. a λ-extension $\lambda(r_i)$ and $\mathcal{P}' = \mathcal{P}_\tau \cup \{r_i'\}$ the program where r_i is replaced by r_i'. Then, $Conflicts(\mathcal{P}') \subseteq Conflicts(\mathcal{P})$.

Proof
Suppose a rule $r_j \in \mathcal{P}$ such that $r_i' \bowtie r_j$ holds. Then by Proposition 2, $B(r_j)$ and $B(r_i')$ do not contain conflict-preventing literals. Since $B(r_i) \subseteq B(r_i')$, this implies that $B(r_j)$ and $B(r_i)$ do not contain conflict-preventing literals either and, therefore, $r_j \bowtie r_i$ holds. Consequently, if a λ-extended rule r_i' w.r.t. a λ-extension $\lambda(r_i)$ is in conflict with another rule $r_j \in \mathcal{P}$, then r_i and r_j are already conflicting. □

Note that Proposition 3 holds for general λ-extensions and not exclusively for conflict-resolving ones.

Extending $B(r_i)$ with a subset-minimal λ-extension $\lambda(r_i)$, therefore, constitutes a *cautious change* in $\mathcal{P}$. As a consequence, a uniformly non-contradictory program core $\mathcal{P}_\mathcal{C}$ is obtained by applying changes to $\mathcal{P}_\mathcal{C}$ that are *justified* in the technical and logical sense.

3.4 A Naïve approach: Semi-normal completion

Preventing the derivation of contradictions is a well-known problem. In default logic for example, *normal default theories* in Reiter (1980) provide a solution to this problem by restricting the form of defaults. In normal default theory, every default has to be of the form $\frac{\alpha:\beta}{\beta}$, meaning that β can only be concluded if α is explicitly true and β can be assumed to be true. This guarantees that the default theory has at least one extension. This idea was adapted for rules of logic programs by Caminada in Caminada (2006), Caminada and Sakama (2006). In Caminada (2006), *semi-normal defeasible rules* are introduced. These rules have the following form:

$$r: L_0 \leftarrow L_1, \dots, L_m, \sim L_{m+1}, \dots, \sim L_n, \sim \overline{L_0}. \tag{5}$$

Obviously, if a program core $\mathcal{P}_{\mathcal{C}}$ only consists of semi-normal defeasible rules, $\mathcal{P}_{\mathcal{C}}$ is also uniformly non-contradictory. The transformation of a program core $\mathcal{P}_{\mathcal{C}}$ to a uniformly non-contradictory core $SN(\mathcal{P}_{\mathcal{C}})$ using semi-normal defeasible rules can, therefore, be defined in a straightforward way. The *semi-normal completion* $SN(\mathcal{P}_{\mathcal{C}})$ of $\mathcal{P}_{\mathcal{C}}$ arises from the extension of the body of each rule $r \in \mathcal{P}_{\mathcal{C}}$ by the literal that is dual to the head literal of r.

Definition 7
Given a rule $r\colon L \leftarrow B(r).$, the *semi-normal completion* $sn(r)$ of r is a rule of the following form:

$$sn(r)\colon L \leftarrow B(r), \sim\overline{L}.$$

Definition 8
Given a program core $\mathcal{P}_{\mathcal{C}}$, the *semi-normal completion* $SN(\mathcal{P}_{\mathcal{C}})$ of $\mathcal{P}_{\mathcal{C}}$ is the program core $SN(\mathcal{P}_{\mathcal{C}}) = \{sn(r) \mid r \in \mathcal{P}_{\mathcal{C}}\}$.

In case there is a conflict in the initial program core $\mathcal{P}_{\mathcal{C}}$, this conflict is prevented by the additional literals in the rule bodies of the conflicting rules which are dual to the respective head literal.

It is easy to see that when using a semi-normal completed program core $\mathcal{P}_{\mathcal{C}}$ over $\mathcal{A}$, the restriction that a valid input $\mathcal{P}_{\mathcal{F}}$ for $\mathcal{P}_{\mathcal{C}}$ can only consist of literals over $\mathcal{E}_{\mathcal{P}}$ becomes void. Instead, it would allow the input $\mathcal{P}_{\mathcal{F}}$ for $\mathcal{P}_{\mathcal{C}}$ to be over $\mathcal{A}$. However, this type of *automated conflict-prevention* does not explicitly resolve the underlying problems, that is, the actual contradictions that are modelled by the rules of the initial $\mathcal{P}_{\mathcal{C}}$.

Example 2
Consider the following program core:

$$SN(\mathcal{P}_{\mathcal{C},2})\colon \quad r_1\colon \overline{allergicToPeanuts} \leftarrow \sim allergicEvent.$$
$$r_2\colon allergicToPeanuts \leftarrow testedPositivePA.$$
$$r_3\colon canEatPeanuts \leftarrow \overline{allergicToPeanuts}.$$

$\mathcal{P}_{\mathcal{C},2}$ states that if someone never had an allergic event, they are not allergic to peanuts. If someone is tested positive for a peanut allergy, they are indeed allergic to it. If someone is explicitly not allergic to peanuts, they are allowed to eat peanuts. The semi-normal completion of $\mathcal{P}_{2,\mathcal{C}}$ yields:

$$SN(\mathcal{P}_{\mathcal{C},2})\colon \quad sn(r_1)\colon \overline{allergicToPeanuts} \leftarrow \sim allergicEvent, \sim allergicToPeanuts.$$
$$sn(r_2)\colon allergicToPeanuts \leftarrow testedPositivePA, \sim\overline{allergicToPeanuts}.$$
$$sn(r_3)\colon canEatPeanuts \leftarrow \overline{allergicToPeanuts}, \sim\overline{canEatPeanuts}.$$

Assume the input $\mathcal{P}_{\mathcal{F}} = \{testedPositivePA.\}$ for $SN(\mathcal{P}_{\mathcal{C},2})$. For $\mathcal{P}_{\mathcal{F}} \cup SN(\mathcal{P}_{\mathcal{C},2})$, we get the answer sets

$$S_1 = \{testedPositivePA, allergicToPeanuts\}, \text{ and}$$
$$S_2 = \{testedPositivePA, \overline{allergicToPeanuts}, canEatPeanuts\}.$$

Clearly, S_2 shows that $\mathcal{P}_{\mathcal{C},2}$ contains wrong information or lacks crucial information. For example, in this case, r_1 only looks at allergic events without considering the possibility that one never consumed a peanut and therefore never had a reaction to begin with.

Semi-normal completions of conflicting rules rather "bypass" potential contradictions as the complementary statements are just considered separately instead of at the same time, which, of course, leads to contradictions. This, in turn, means that given a program core $\mathcal{P}_\mathcal{C}$ with conflicting rules, solving programs $\mathcal{P} \in \Pi(SN(\mathcal{P}_\mathcal{C}))$ with the semi-normal completed program core will in fact yield answer sets that are not adequate in the professional sense as at least one of the answer sets will represent unintended conclusions as we have seen in Example 2. From a functional perspective, such an effect is not desirable whatsoever. Semi-normal completions can, therefore, be in fact regarded as quickfixes in order to assure consistent answer sets regardless of what the resulting answer sets actually represent. Example 2 illustrates that semi-normal program cores do not suffice if one wants to establish a knowledge base that represents adequate information and their relations in the professional sense. To incrementally *repair* the knowledge base, one has to analyze conflicting rules and explicitly add, remove, or modify rules in order to get rid of all conflicts.

3.5 Conflict resolution with informative extensions

In the following, we present an approach to obtain uniformly non-contradictory program cores by computing conflict-resolving λ-extensions for rules r_i. Since λ-extensions only comprise literals whose atoms occur in the bodies of the adversaries $r_j \in Adv(r_i)$ and that do not contain literals that are atom-related to $H(r_i)$, the extensions are *informative* in the sense that they utilize the body literals of the rules that are involved in the conflicts of r_i without introducing new atoms or using the head literal. Semi-normal completions are, in comparison, a purely technical device to prevent contradictions and thus not informative.

According to Theorem 1, for every rule $r_j \in Adv(r_i)$, $\lambda(r_i)$ has to contain at least one conflict-preventing literal, that is, an extended literal L^* such that there exists an extended literal $K^* \in B(r_j)$ and L^*, K^* are complementary. This means that a conflict-resolving λ-extension for r_i can be obtained by finding a set β of literals that contains at least one body literal of each rule $r_j \in Adv(r_i)$ and that does not contain any literals that occur in the body of r_i. This set β can then be transformed into a conflict-resolving λ-extension $\lambda(r_i)$ by negating each literal in this set. It is easy to see that in order for the λ-extended rule r_i' to remain applicable, each computed conflict-resolving λ-extension and therefore each such β has to meet additional constraints which we will explore next.

In order to compute these sets β, we will first adapt the notion of *hitting sets* (Berge 1989) as follows: Let $\mathcal{AB}(r_i) = \{B(r_j) \mid r_j \in Adv(r_i)\}$ be the collection of all rule bodies in $\mathcal{P}_\mathcal{C}$ whose rules are in conflict with r_i. A *hitting set* of $\mathcal{AB}(r_i)$ is a subset $h \subseteq Lit_\mathcal{A}^*$ that meets every set in $\mathcal{AB}(r_i)$, that is, $B(r_j) \cap h \neq \emptyset$ for every $B(r_j) \in \mathcal{AB}(r_i)$. A hitting set h is *minimal* if there does not exist a proper subset of h that is also a hitting set. It is easy to see that minimal hitting sets of $\mathcal{AB}(r_i)$ provide a good starting point to attain suitable conflict-resolving extensions $\lambda(r_i)$ since every hitting set shares at least one extended literal with the body of every rule $r_j \in Adv(r_i)$. One could assume that to compute a set of conflict-preventing literals, it suffices to just take any minimal hitting set

of $\mathcal{AB}(r_i)$ and simply default- or strongly negate each extended literal of that hitting set. But as mentioned before, not every hitting set can be used to compute conflict-resolving extensions as there are some more conditions that have to be met. We have to ensure that a λ-extended rule r_i' of r_i is still applicable. The applicability is not given if (a) the conflict-resolving extension itself is already inconsistent or (b) the extended rule body $B(r_i')$ becomes inconsistent. Case (a) applies if a hitting set h contains complementary or reconcilable literals L^*, K^* as every combination of their strongly and default negated complement is also complementary.[2] To ensure that (b) does not apply, we only consider those hitting sets that do not comprise literals that are atom-related to literals in $B(r_i)$. Hence, hitting sets that contain complementary or reconcilable literals and hitting sets that share common underlying atoms with $B(r_i)$ will not be used to compute conflict-resolving λ-extensions.

Remark 2: The restriction emerging from (b) is a bit stricter than necessary as rules with atom-related literals in their bodies are still satisfiable if these atom-related literals are compatible. But in order to avoid unnecessary technicalities, we will omit this special case here.

All these restrictions lead us to the following extension of hitting sets called *blankets* which we will use to compute proper conflict-resolving λ-extensions:

Definition 9 (*Blanket*)
A *blanket* β for $Adv(r_i)$ is a non-empty, $\subseteq$-minimal, consistent set of extended literals $L^* \in Lit_{\mathcal{A}}^*$ without reconcilable literals such that $Atom(\beta) \cap Atom(B(r_i)) = \emptyset$ and for each $r_j \in Adv(r_i)$, there exists an extended literal $L^* \in \beta$ with $L^* \in B(r_j)$. We denote the set of all blankets for $Adv(r_i)$ by $blankets(Adv(r_i))$.

Example 3
Suppose the following rules:

$$r_1\colon a \leftarrow b, c, d.$$
$$r_2\colon \overline{a} \leftarrow b, c.$$
$$r_3\colon \overline{a} \leftarrow b, \sim e.$$
$$r_4\colon \overline{a} \leftarrow b, \sim \overline{e}.$$
$$r_5\colon \overline{a} \leftarrow b, e.$$

Consider the following ELP: $\mathcal{P}_3 = \{r_1, r_2\}$. For $Adv(r_1)$ in $\mathcal{P}_3$, there does not exist a blanket since every body literal of r_2 also occurs in the body of r_1.

Now, consider the ELP $\mathcal{P}_3' = \{r_1, r_3, r_4\}$. The only body literals of r_3, r_4 that do not occur in r_1 are $\sim e, \sim \overline{e}$. Since a blanket for $Adv(r_1)$ in $\mathcal{P}_3'$ must not contain reconcilable literals, there does not exist a blanket for $Adv(r_1)$.

Likewise, for the ELP $\mathcal{P}_3'' = \{r_1, r_3, r_5\}$, we observe the following: The only body literals of r_3, r_5 that do not occur in r_1 are $\sim e, e$. Since a blanket for $Adv(r_1)$ in $\mathcal{P}_3''$ must not contain complementary literals, there does not exist a blanket for $Adv(r_1)$.

---

[2] Keep in mind that default literals cannot be strongly negated, for example, $\sim\overline{L}, \sim K$ are not negated counterparts for two atom-related literals $\sim L, \sim \overline{K}$.

Example 4

Suppose an ELP $\mathcal{P}_4$ with the following rules:

$$r_1\colon eligX \leftarrow condAAdv.$$
$$r_2\colon \overline{eligX} \leftarrow \sim highLCount, \sim preTreatedN.$$
$$r_3\colon \overline{eligX} \leftarrow \overline{highLCount}, \sim ctrIndR.$$
$$r_4\colon \overline{eligX} \leftarrow highLCount, \sim preTreatedM.$$

Program $\mathcal{P}_4$ describes the following scenario: Suppose a therapy X that was developed for patients with condition A. Currently only patients with an advanced form of condition A are eligible (r_1). Furthermore, imagine a specific laboratory value L in a patient's body that can only be determined via an invasive test which is why such a test can only be done once every 6 months at most. Recently completed long-term studies now indicate that there are some exceptions where X must not be recommended to a patient with advanced A. The therapy should not be used on a patient if it is unknown whether they currently have a high or low L-count and if it is also unknown if they were treated with drug N sometime in the past (r_2). If the usage of a substance R is not contraindicated andthe patient has a low L-count, treatment X must also not be recommended (r_3). Finally, if patient has a high L-count, they should not receive treatment X if it cannot be concluded that the patient was treated with drug M in the past (r_4).

Rule r_1 is in conflict with every other rule:

$$conflicts(r_1) = \{(r_1, r_2), (r_1, r_3), (r_1, r_4)\}$$
$$Adv(r_1) = \{r_2, r_3, r_4\}$$

The literals of the rule bodies in $Adv(r_1)$ do not share any common underlying atoms with the literals in the body of r_1. The set $blankets(Adv(r_1))$ of all blankets for $Adv(r_1)$ consists of the sets $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5$ where

$$\beta_1 = \{\sim highLCount, \overline{highLCount}, \sim preTreatedM\},$$
$$\beta_2 = \{\sim highLCount, \sim ctrIndR, \sim preTreatedM\},$$
$$\beta_3 = \{\overline{highLCount}, \sim preTreatedN, \sim preTreatedM\},$$
$$\beta_4 = \{\sim preTreatedN, \sim ctrIndR, \sim preTreatedM\}, \text{ and}$$
$$\beta_5 = \{highLCount, \sim preTreatedN, \sim ctrIndR\}.$$

For the remainder of this paper, we will assume that for any rule r_i with $Adv(r_i) \neq \emptyset$, there exists at least one blanket for $Adv(r_i)$. This particularly implies that for any given conflict pair (r_i, r_j), $Atom(B(r_j)) - Atom(B(r_i)) \neq \emptyset$ has to hold. We will assume that r_i will be chosen such that this condition is satisfied. If this is not possible in practice, one or both rules have to be modified more individually, for example, by introducing new atoms to $\mathcal{P}$. Other solutions will be discussed in Section 3.6. Now, we will describe how to compute all possible conflict-resolving λ-extensions $\lambda(r_i)$ given a blanket β for $Adv(r_i)$.

In order to use a blanket $\beta \in blankets(Adv(r_i))$ to resolve all conflicts (r_i, r_j) in $conflicts(r_i)$, the literals in β have to be negated such that for each conflict pair, the addition of the negated literals to the body of r_i leads to the violation of either property (CP2) or (CP3) in Theorem 1. Due to the fact that default and strong negation cannot

be used interchangeably as default literals cannot be strongly negated, we propose the following *neg*-operator, which for an extended literal L^* with $atom(L^*) = A$, outputs the set of literals that are complementary to L^*:

$$neg(L^*) = \begin{cases} \{\sim A, \overline{A}\} & \text{if } L^* = A \\ \{\sim \overline{A}, A\} & \text{if } L^* = \overline{A} \\ \{A\} & \text{if } L^* = \sim A \\ \{\overline{A}\} & \text{if } L^* = \sim \overline{A}. \end{cases} \tag{6}$$

Therefore, the *neg*-operator can be used to compute the proper conflict-preventing literals. In fact, we want to extend the *neg*-operator such that the application of the resulting operator on a blanket for r_i will yield the set of all corresponding conflict-resolving λ-extensions $\lambda(r_i)$. For this purpose, we have to ensure that the negations of all literals in a blanket yield extended rules of the form (4) that are still applicable. Consequently a blanket must not contain complementary literals. Dual literals however are allowed in a blanket since their negation via the *neg*-operator leads to non-complementary literals.

Additionally, since we want minimal sets of conflict-resolving literals, given a blanket with dual literals L^*, K^*, we only want to consider those negated forms that both L^*, K^* have in common. Therefore, we extend our *neg*-operator such that the negation of atom-related literals only yields the negated form that both literals have in common, that is,

$$neg_X(L^*) = \cap \{neg(K^*) \mid atom(K^*) = atom(L^*), K^* \in X\}, \tag{7}$$

where X is a set of extended literals. Applied on a blanket and their literals, the resulting literals uphold a crucial property:

Corollary 2
Given a blanket β, (a) for all atom-related literals $L^*, K^* \in \beta$, $neg_\beta(L^*) = neg_\beta(K^*)$, and (b) for all other literals L^*, K^*, $neg_\beta(L^*) \cap neg_\beta(K^*) = \emptyset$.

Example 5
Suppose the following sets $X_1 = \{a, \overline{b}, \sim b\}$, $X_2 = \{a, b, \overline{b}\}$, and $X_3 = \{a, b, \sim b\}$. The negation of these sets via the *neg*-operator yields

$$neg_{X_1}(a) = neg_{X_2}(a) = neg_{X_3}(a) = \{\overline{a}, \sim a\},$$
$$neg_{X_1}(\overline{b}) = neg_{X_1}(\sim b) = \{b, \sim \overline{b}\} \cap \{b\} = \{b\},$$
$$neg_{X_2}(b) = neg_{X_2}(\overline{b}) = \{\sim b, \overline{b}\} \cap \{\sim \overline{b}, b\} = \emptyset, \text{ and}$$
$$neg_{X_3}(b) = neg_{X_3}(\sim b) = \{\overline{b}, \sim b\} \cap \{b\} = \emptyset.$$

The negation of dual literals in X_1 leads to the output of the common literal b whereas complementary literals like in X_2 and X_3 do not share common negated forms.

Example 5 illustrates why blankets are allowed to contain dual literals since they share a common negated form, but must not have complementary literals. When looking at the negation of X_2 in Example 5, one can see that strongly negated literals still have negated forms that are not complementary, viz., the reconcilable literals $\sim b, \sim \overline{b}$. In contrast to complementary literals, reconcilable literals in a rule body will not inhibit the satisfiability of the rule. The examples, therefore, demonstrate that by requesting common negated

forms, our approach is prima facie stricter than necessary. But in order to keep this approach simple while guaranteeing that conflict-resolving λ-extensions maintain the satisfiability of the extended rule, we will only allow combinations of atom-related literals whose negations are guaranteed to be non-complementary.

Lemma 1
Let an ELP $\mathcal{P}$ and a rule $r \in \mathcal{P}$ be given. For each extended literal L^* in a blanket β for $Adv(r)$, $neg_\beta(L^*)$ is a non-empty set of extended literals which are complementary to L^*.

Proof
Since the *neg*-operator as defined in (6) returns a non-empty set for any extended literal L^*, $neg_\beta(L^*)$ can only be an empty set if β contains some distinct atom-related literals $K_1^*, \ldots, K_m^*$ such that $neg(K_1^*) \cap \cdots \cap neg(K_m^*) = \emptyset$. Suppose atom-related literals K_1^*, K_2^*, K_3^*. By (6), $neg(K_1^*) \cap neg(K_2^*) \neq \emptyset$ iff K_1^*, K_2^* are dual. In all other cases, $neg(K_1^*) \cap neg(K_2^*) = \emptyset$. Let therefore K_1^*, K_2^* be dual literals. For any K_3^*, it is easy to see that by (6) either $neg(K_1^*) \cap neg(K_3^*) = \emptyset$ or $neg(K_2^*) \cap neg(K_3^*) = \emptyset$ holds. This is due to the fact that in every set of at least three different atom-related literals, there are at least two literals that are not dual. Consequently, $neg_\beta(L^*) = \emptyset$ if β contains more than two atom-related literals or if there exist two atom-related literals $K_1^*, K_2^* \in \beta$ such that K_1^*, K_2^* are not dual. A blanket does not contain complementary or reconcilable literals by definition and only allows atom-related literals that are dual. This implies that for each extended literal K_1^* in β, there exists at most one other extended literal K_2^* in β such that K_1^*, K_2^* are atom-related, and all atom-related literals in β are dual. Thus, for each extended literal $L^* \in \beta$, $neg_\beta(L^*)$ is not empty. More precisely, $neg_\beta(L^*)$ always contains either $\overline{L}$ or $\sim L$ or both. If β does not contain literals that are atom-related to L^*, then $neg_\beta(L^*) = neg(L^*)$ holds. $\qquad\square$

For a set X of extended literals L^*, we define $Neg(X)$ as the set of all possible sets T where T contains every literal of X in a negated form, that is,

$$Neg(X) = \{T \mid T \subseteq \mathcal{L}^*(X), \ T \text{ contains exactly one element of} \\ \text{each non-empty set } neg_X(L^*), L^* \in X\}, \tag{8}$$

where $\mathcal{L}^*(X) = \bigcup_{L^* \in X} neg_X(L^*)$. Due to Lemma 1, there always exists at least one such set T.

Example 6 (Example 5 continued)
Continuing Example 5, we get $Neg(X_2) = Neg(X_3) = \big\{\{\overline{a}\}, \{\sim a\}\big\}$ since $neg_{X_2}(a) = neg_{X_3}(a) = \{\overline{a}, \sim a\}$, and the negation of all other literals yields an empty set. For X_1, however, we get $Neg(X_1) = \big\{\{\overline{a}, b\}, \{\sim a, b\}\big\}$ because as for X_2, X_3, the negation of a produces two different possible negations and additionally $\overline{b}$ and $\sim b$ have the common negated form b.

Example 7 (Example 4 continued)
Continuing Example 4, the negation of each blanket in $blankets(Adv(r_1))$ for rule r_1 in $\mathcal{P}_4$ yields sets $\beta_1, \ldots, \beta_5$ as shown in Table 2. Blankets β_2 and β_4 only contain default literals and therefore, each of them has only one possible negation form. For β_3 and

Table 2. *Results from Example 7*

β_i	$neg_{\beta_i}(L^*)$	$Neg(\beta_i)$
β_1	$\{highLCount\}, \{preTreatedM\}$	$\{\{highLCount, preTreatedM\}\}$
β_2	$\{highLCount\}, \{ctrIndR\},$ $\{preTreatedM\}$	$\{\{highLCount, ctrIndR, preTreatedM\}\}$
β_3	$\{highLCount, \sim\overline{highLCount}\},$ $\{preTreatedN\}, \{preTreatedM\}$	$\{\{highLCount, preTreatedN, preTreatedM\},$ $\{\sim\overline{highLCount}, preTreatedN, preTreatedM\}\}$
β_4	$\{preTreatedN\}, \{ctrIndR\},$ $\{preTreatedM\}$	$\{\{preTreatedN, ctrIndR, preTreatedM\}\}$
β_5	$\{\overline{highLCount}, \sim highLCount\},$ $\{preTreatedN\}, \{ctrIndR\}$	$\{\{\overline{highLCount}, preTreatedN, ctrIndR\},$ $\{\sim highLCount, preTreatedN, ctrIndR\}\}$

β_5, the literals $highLCount$, and $\overline{highLCount}$, each have two possible negations which is why $Neg(\beta_3)$ and $Neg(\beta_5)$ contain two sets respectively. Regarding β_1, we observe that the dual literals $\overline{highLCount}, \sim highLCount$, have the common negated form $highLCount$, and the default literal $\sim preTreatedM$ has only one negation form. Together, that results in the fact that $Neg(\beta_1)$ contains only the one set $\{highLCount, preTreatedM\}$.

Lemma 2
Let an ELP $\mathcal{P}$, a rule $r \in \mathcal{P}$ and a blanket β for $Adv(r)$ be given. Furthermore, let $\lambda \in Neg(\beta)$. For each extended literal $L^* \in \beta$, λ contains exactly one extended literal K^* such that L^*, K^* are complementary.

Proof
Lemma 1 states that for each $L^* \in \beta$, $neg_\beta(L^*)$ consists of either one or two extended literals and each of them is complementary to L^*. Consequently, it follows by (8) that each λ contains a literal K^* for each L^* in β such that L^*, K^* are complementary. With Corollary 2, there exists exactly one such extended literal $K^* \in \lambda$ for each $L^* \in \beta$. $\square$

Due to the fact that $blankets(Adv(r_i))$ is a set of sets, we will extend the Neg-operator to also consider sets $\mathcal{X} = \{X_1, \ldots, X_n\}$ of sets $X_1, \ldots, X_n$ of extended literals. For such a set $\mathcal{X}$, $N(\mathcal{X})$ shall therefore output the set of all negated form variations in all sets $X_1, \ldots, X_n \in \mathcal{X}$, that is,

$$N(\mathcal{X}) = \{X^- \mid X^- \in Neg(X), X \in \mathcal{X}\}. \tag{9}$$

Lemma 3
Let an ELP $\mathcal{P}$ and a rule $r \in \mathcal{P}$ be given. Each $\lambda \in N(blankets(Adv(r)))$ does not contain any complementary literals.

Proof
Since every λ consists of a literal $L^* \in neg_\beta(L^*)$ for every literal $L^* \in \beta$, λ can only contain complementary literals if there are complementary literals between sets $neg_\beta(L^*)$. By Corollary 2, this is not possible since β itself does not contain complementary or

reconcilable literals. Consequently, for each $\lambda \in N(blankets(Adv(r)))$, it holds that λ does not contain complementary literals. $\qquad\square$

Example 8 (*Example 7 continued*)
For $blankets(Adv(r_1))$ in Example 4, we get

$$N(blankets(Adv(r_1))) = \{Neg(\beta_1), Neg(\beta_2), Neg(\beta_3), Neg(\beta_4), Neg(\beta_5)\}$$
$$= \{\{highLCount, preTreatedM\},$$
$$\{highLCount, ctrIndR, preTreatedM\},$$
$$\{highLCount, preTreatedN, preTreatedM\},$$
$$\{\sim\overline{highLCount}, preTreatedN, preTreatedM\},$$
$$\{preTreatedN, ctrIndR, preTreatedM\},$$
$$\{\overline{highLCount}, preTreatedN, ctrIndR\},$$
$$\{\sim highLCount, preTreatedN, ctrIndR\}\}.$$

Due to the properties of a blanket and their possible negations, it is also guaranteed that any rule extended by such a λ remains applicable.

Proposition 4
Every λ-extended rule r_i' of the form

$$r_i' \in \{H(r_i) \leftarrow B(r_i), \lambda(r_i). \mid \lambda(r_i) \in N(blankets(Adv(r_i)))\} \qquad (10)$$

of a rule $r_i \in \mathcal{P}$ is applicable.

Proof
Suppose a λ-extended rule r_i' w.r.t. a λ-extension $\lambda(r_i)$. Rule r_i' is not applicable if (a) r_i is not applicable, (b) $\lambda(r_i)$ is not satisfiable, or (c) $B(r_i) \cup \lambda(r_i)$ is not satisfiable. Case (a) is not possible due to our initial assumption that all rules in a given logic program are applicable. Case (b) can only hold if $\lambda(r_i)$ contains complementary literals. By Lemma 3, this is not possible for any conflict-resolving extension $\lambda(r_i)$. Then, case (c) can only hold if there exist complementary literals between $B(r_i)$ and $\lambda(r_i)$. By Definition 9, a blanket for r_i cannot contain literals that are atom-related to any literals in $B(r_i)$. Consequently, any conflict-resolving extension $\lambda(r_i)$ for r_i does also not contain any literals that are atom-related to a literal in $B(r_i)$. Therefore, any λ-extended rule of the form (10) is applicable. $\qquad\square$

We are now ready to relate conflict-resolving λ-extensions to blankets.

Proposition 5
Every $\lambda(r_i) \in N(blankets(Adv(r_i)))$ is a conflict-resolving λ-extension for r_i, that is, for every rule λ-extended rule r_i' w.r.t. $\lambda(r_i)$, it holds that $conflicts(r_i') = \emptyset$ in $\mathcal{P}' = \mathcal{P}\backslash\{r_i\} \cup \{r_i'\}$.

Proof
Assume, by way of contradiction, that there exists a rule $r_j \in Adv(r_i)$ such that the λ-extended rule r_i' w.r.t. $\lambda(r_i)$ and r_j are conflicting, that is, $r_i' \bowtie r_j$. Then by Theorem 1, (CP2) and (CP3) hold, that is, $B^+(r_i') \cap B^-(r_j) = B^+(r_j) \cap B^-(r_i') = \emptyset$, and $B^+(r_i') \cup$

$B^-(r_j)$ and $B^-(r_i') \cup B^+(r_j)$ are consistent. Let β be a blanket on which $\lambda(r_i)$ is based on, that is, $\beta \in blankets(Adv(r_i))$ and $\lambda(r_i) \in Neg(blankets(Adv(r_i)))$. According to (9), this also means that $\lambda(r_i) \in Neg(\beta)$ holds. By Definition 9, there exists an extended literal $L^* \in \beta$ such that $L^* \in B(r_j)$. If $L \in B^+(r_j)$, then by Lemma 2 either $\overline{L} \in \lambda(r_i)$ or $\sim L \in \lambda(r_i)$. This consequently means that if $L \in B^+(r_j)$, then either $\overline{L} \in B^+(r_i')$, which means condition (CP3) cannot hold or $L \in B^-(r_i')$, which in turn means that (CP2) cannot hold. On the other hand, if $L \in B^-(r_j)$, then by Lemma 2 and (4) $L \in B^+(r_i')$, and thus (CP2) cannot hold. In any case, $\lambda(r_i)$ adds an extended literal to the body of r_i such that (CP2) or (CP3) cannot hold for rules r_i', r_j. Therefore, r_i' and r_j cannot be conflicting. □

Example 9 (*Example 8 continued*)
All conflicts in $Conflicts(\mathcal{P}_4)$ of Example 4 can be resolved by replacing r_1 with a λ-extended rule

$$r_1': \, eligX \leftarrow condAAdv, \lambda(r_1).$$

w.r.t. $\lambda(r_1)$, where

$$\lambda(r_1) \in N(blankets(Adv(r_1))) = \big\{\{highLCount, preTreatedM\},$$
$$\{highLCount, ctrIndR, preTreatedM\},$$
$$\{highLCount, preTreatedN, preTreatedM\},$$
$$\{\sim\overline{highLCount}, preTreatedN, preTreatedM\},$$
$$\{preTreatedN, ctrIndR, preTreatedM\},$$
$$\{\overline{highLCount}, preTreatedN, ctrIndR\},$$
$$\{\sim highLCount, preTreatedN, ctrIndR\}\big\}.$$

Every set in $N(blankets(Adv(r_1)))$ contains at least one body literal of every rule $r_j \in Adv(r_1)$ in a negated form and therefore corresponds to a conflict-resolving λ-extension for r_1 in the sense of (4).

Looking at the result in Example 8, it becomes apparent that the N-operator for sets of sets does not necessarily output $\subseteq$-minimal sets, that is, $N(blankets(Adv(r_1)))$ contains $\{highLCount, preTreatedM\}$ and two of its supersets $\{highLCount \, ctrIndR, preTreatedM\}$ and $\{highLCount, preTreatedN, preTreatedM\}$. Since $\{highLCount, preTreatedM\}$ already contains literals of every adversarial rule of r_1 in a negated form, additional literals become obsolete. Consequently, any superset of a set in $N(blankets(Adv(r_1)))$ can be safely omitted. Therefore, for an ELP $\mathcal{P}$ with a rule r_i, we denote the set of all $\subseteq$-minimal sets in $N(blankets(Adv(r_i)))$ by $N_{\min}(blankets(Adv(r_i)))$.

Corollary 3
All conflicts in $conflicts(r_i)$ are resolved simultaneously if r_i is replaced by

$$r_i' \in \{H(r_i) \leftarrow B(r_i), \lambda(r_i). \mid \lambda(r_i) \in N_{\min}(blankets(Adv(r_i)))\}. \tag{11}$$

Every $\lambda(r_i) \in N_{\min}(blankets(Adv(r_i)))$ is, therefore, a *minimal* conflict-resolving λ-extension for r_i.

Example 10 (*Example 9 continued*)

For r_1 in $\mathcal{P}_4$ of Example 4, we get

$$N_{\min}(blankets(Adv(r_1))) = \big\{\{highLCount, preTreatedM\},$$
$$\{\sim\overline{highLCount}, preTreatedN, preTreatedM\},$$
$$\{preTreatedN, ctrIndR, preTreatedM\},$$
$$\{\overline{highLCount}, preTreatedN, ctrIndR\},$$
$$\{\sim highLCount, preTreatedN, ctrIndR\}\big\}.$$

Consequently, $\lambda_1 = \{highLCount, preTreatedM\}$ is a possible λ-extension for r_1.

Note that if multiple conflicts are resolved simultaneously where the adversarial rules contain more than one literal, the inclusion of a knowledge expert can be crucial. In Example 10, any possible λ-extension for r_1 proposes a restriction on the eligibility for therapy X that is slightly stricter than the requirements that are implictly imposed by the adversarial rules. That is, the extension by λ_1 leads indeed to the resolution of the conflict. Now however, for instance patients that have a low L-count while being allergic to substance R cannot receive the treatment which is not explicitly specified in the original program. One reason for these stricter suggestions is that λ-extension are built to minimally cover the adversarial body literals in order to keep the approach simple and pragmatic. It is up to the knowledge expert to decide if the new knowledge that is represented in a λ-extension is justifiable w.r.t. to its professional adequacy. If the knowledge expert wants to adapt r_1 according to all adversarial rules such that r_1 "mirrors" them, they can consider accepting more than one suggestion which in this case would result in replacing r_1 by multiple λ-extended rules r_1' that guarantee that specific literal combinations are not overlooked, viz., an additional λ-extended rule of r_1 would have to be added where $\lambda = \{\overline{highLCount}, preTreatedN, ctrIndR\}$ if the knowledge expert wants patients with a low L-count to receive treatment X where substance R is contraindicated.

Once all λ-extensions for a conflict are computed, the corresponding λ-extended rules are presented to the expert. They can analyze the suggestions and apply the most suitable solution. Due to their expertise, the presented suggestions can hint to the underlying cause which can make the decision on the solution straightforward.

Moreover, though it demands for some technical knowledge, the framework could also allow the expert to refine a suggestion and then apply it. Even though, as shown above, all computed solutions are minimal and guarantee the resolution of the considered conflict, refining a solution based on the expert's knowledge can reinforce the robustness of the solution, that is, possible future conflicts after consecutive updates can be prevented.

Example 11 (*Example 10 continued*)

Suppose the knowledge expert has to decide on the solution for the conflict of rule r_1. They are presented with the following suggestions:

$$r_1^{(1)}\colon eligX \leftarrow highLCount, preTreatedM.$$
$$r_1^{(2)}\colon eligX \leftarrow \sim\overline{highLCount}, preTreatedN, preTreatedM.$$
$$r_1^{(3)}\colon eligX \leftarrow preTreatedN, ctrIndR, preTreatedM.$$
$$r_1^{(4)}\colon eligX \leftarrow \overline{highLCount}, preTreatedN, ctrIndR.$$
$$r_1^{(5)}\colon eligX \leftarrow \sim highLCount, preTreatedN, ctrIndR.$$

Presume the following scenario: Treatment X is applied when the therapy options for advanced condition A that were administered up to now has been exhausted. Contrary to treatment M, which is a more common therapy in this scenario, treatment N is a therapy that is recommended when the standard therapy options have not been effective since N is known to be more complicated and not without risks. Therefore, the knowledge expert decides that the solution for this conflict should include *preTreatedN* rather than *preTreatedM* to reflect that therapy X is only to be recommended if nothing else worked out. This leaves the expert with rules $r_1^{(4)}$ and $r_1^{(5)}$ as potential candidates. As therapy X itself is also not without risks, the expert wants to emphasize that any other requirements for this therapy should be defined as strictly as possible, for example, the physicians have to explicitly confirm that specific drugs are not contraindicated. Accordingly, before suggesting therapy X, it should be explicitly known whether the patient has indeed a low L-count. Therefore, rule $r_1^{(4)}$ as a solution is the most fitting candidate.

Example 11 highlights the fact that generally the suitability of a solution relies on knowledge that is not necessarily reflected in the program.

The following example illustrates that even in cases where a custom solution is required, the computed solutions can used as a basis for fitting solutions.

Example 12 (*Example 11 continued*)
Now suppose that at the time of the conflict resolution, studies revealed that in order to prescribe treatment X, the patient must not have an allergy towards a substance S. As $r_1^{(4)}$ was found to be the most suitable suggestion, it suffices to add literal $\overline{ctrIndS}$ to $B(r_1^{(4)})$ in order to require that substance S is not contraindicated. Thus, the expert yields

$$eligX \leftarrow \overline{highLCount}, preTreatedN, ctrIndR, \overline{ctrIndS}.$$

as the final solution to solve all conflicts of r_1 in $\mathcal{P}_4$. This modification constitutes a refinement of a suggested λ-extension.

In Sections 4.3 and 4.4, we will show that the problem of knowledge gaps will also pertain to other aspects of conflict resolution. We propose to alleviate these shortcomings by implementing suitable interaction mechanisms between the system and the expert.

In the following, we will illustrate a complete conflict resolution step with two more examples. Example 13 comprises both short and long conflict-resolving extensions. Example 14 shows how redundant extensions are removed via $N_{\min}$.

Example 13
Suppose the following ELP $\mathcal{P}_{13}$:

$$r_1\colon a \leftarrow x, y.$$
$$r_2\colon \overline{a} \leftarrow b, d, \sim e.$$
$$r_3\colon \overline{a} \leftarrow d, \sim c, \sim e.$$
$$r_4\colon \overline{a} \leftarrow b, d, \sim c.$$

In $\mathcal{P}_{13}$, rule r_1 is in conflict with every other rule:

$$Conflicts(\mathcal{P}_{13}) = conflicts(r_1) = \{(r_1, r_2), (r_1, r_3), (r_1, r_4)\}$$
$$Adv(r_1) = \{r_2, r_3, r_4\}.$$

The set $blankets(Adv(r_1))$ of blankets for $Adv(r_1)$ consists of the sets $\beta_1, \beta_2, \beta_3, \beta_4$, with

$$\beta_1 = \{d\},$$
$$\beta_2 = \{b, \sim c\},$$
$$\beta_3 = \{b, \sim e\},$$
$$\beta_4 = \{\sim c, \sim e\}.$$

The negation of each blanket in $blankets(Adv(r_1))$ results in the following sets:

$$Neg(\beta_1) = \{\{\bar{d}\}, \{\sim d\}\}$$
$$Neg(\beta_2) = \{\{\bar{b}, c\}, \{\sim b, c\}\}$$
$$Neg(\beta_3) = \{\{\bar{b}, e\}, \{\sim b, e\}\}$$
$$Neg(\beta_4) = \{\{c, e\}\}.$$

Therefore, all conflicts in $conflicts(r_1)$ can be resolved by replacing r_1 with a rule

$$r_1': a \leftarrow x, y, \lambda(r_1),$$

where

$$\lambda(r_1) \in N(blankets(Adv(r_1))) = N_{\min}(blankets(Adv(r_1)))$$
$$= \{\{\bar{d}\}, \{\sim d\}, \{\bar{b}, c\}, \{\sim b, c\}, \{\bar{b}, e\}, \{\sim b, e\}, \{c, e\}\}.$$

From a technical point of view, choosing $\lambda(r_1) \in \{\bar{d}, \sim d\}$ as a conflict-resolving extension for r_1 may seem as the most suitable choice prima facie, seeing that the addition of a single literal resolves all three conflicts of r_1 simultaneously. However, depending on what the rules actually represent, it is possible that a corresponding knowledge expert does not regard $\bar{d}$ and $\sim d$ as viable options for resolving the conflicts of r_1, and, instead, picks an extension that contains more than one literal.

Example 14
Suppose the following ELP $\mathcal{P}_{14}$:

$$r_1: x \leftarrow a.$$
$$r_2: \bar{x} \leftarrow \bar{c}, d, \sim \bar{a}, \sim e.$$
$$r_3: \bar{x} \leftarrow a, b, \bar{c}.$$
$$r_4: \bar{x} \leftarrow d, \sim \bar{b}, \sim f.$$

In $\mathcal{P}_{14}$, rule r_1 is in conflict with every other rule:

$$Conflicts(\mathcal{P}_{14}) = conflicts(r_1) = \{(r_1, r_2), (r_1, r_3), (r_1, r_4)\}$$
$$Adv(r_1) = \{r_2, r_3, r_4\}.$$

The set $blankets(Adv(r_1))$ of blankets for $Adv(r_1)$ consists of the sets

$$\beta_1 = \{\bar{c}, d\},$$
$$\beta_2 = \{\bar{c}, \sim \bar{b}\},$$
$$\beta_3 = \{\bar{c}, \sim f\},$$
$$\beta_4 = \{b, d\},$$

$$\beta_5 = \{b, \sim\overline{b}, \sim e\}, \text{ and}$$
$$\beta_6 = \{b, \sim e, \sim f\}.$$

Note that literal $\sim\overline{a}$ does not appear in any blankets of r_1 by reason of $atom(\sim\overline{a}) \in Atom(B(r_1))$.

The negation of each blanket in $blankets(Adv(r_1))$ yields the following sets:

$$Neg(\beta_1) = \big\{\{c, \overline{d}\}, \{\sim\overline{c}, \overline{d}\}, \{c, \sim d\}, \{\sim\overline{c}, \sim d\}\big\}$$
$$Neg(\beta_2) = \big\{\{c, \overline{b}\}, \{\sim\overline{c}, \overline{b}\}\big\}$$
$$Neg(\beta_3) = \big\{\{c, f\}, \{\sim\overline{c}, f\}\big\}$$
$$Neg(\beta_4) = \big\{\{\overline{b}, \overline{d}\}, \{\sim b, \overline{d}\}, \{\overline{b}, \sim d\}, \{\sim b, \sim d\}\big\}$$
$$Neg(\beta_5) = \big\{\{\overline{b}, e\}\big\}$$
$$Neg(\beta_6) = \big\{\{\overline{b}, e, f\}, \{\sim b, e, f\}\big\}.$$

This results in

$$N(blankets(Adv(r_1))) = \bigcup_{\beta \in blankets(Adv(r_1))} Neg(\beta),$$

and on the grounds that $\{\overline{b}, e, f\} \supset \{\overline{b}, e\}$, we get

$$N_{\min}(blankets(Adv(r_1))) = N(blankets(Adv(r_1))) \backslash \{\overline{b}, e, f\}.$$

Therefore, all conflicts in $conflicts(r_1)$ can be resolved by replacing r_1 with a rule

$$r_1'\colon x \leftarrow a, \lambda(r_1),$$

where

$$\lambda(r_1) \in N_{\min}(blankets(Adv(r_1))) = \big\{\{c, \overline{d}\}, \{\sim\overline{c}, \overline{d}\}, \{c, \sim d\}, \{\sim\overline{c}, \sim d\}\{c, \overline{b}\}, \{\sim\overline{c}, \overline{b}\},$$
$$\{c, f\}, \{\sim\overline{c}, f\}, \{\overline{b}, \overline{d}\}, \{\sim b, \overline{d}\}, \{\overline{b}, \sim d\}, \{\sim b, \sim d\},$$
$$\{\overline{b}, e\}, \{\sim b, e, f\}\big\}.$$

In summary, the results show that implementing a conflict resolution step via conflict-resolving λ-extensions and blankets yields a conflict resolution process that possesses the properties (P1)-(P3) that were postulated earlier. By Proposition 5, given a rule $r_i \in \mathcal{P}$, every $\lambda(r_i) \in N(blankets(Adv(r_i)))$ is a conflict-resolving λ-extension for r_i. Since by every such conflict resolution step, the number of conflicts is reduced, for the final program $\mathcal{P}^{(n)}$ of the conflict resolution process, it holds that $Conflicts(\mathcal{P}^{(n)}) = \emptyset$. Moreover, we have shown that for the proposed conflict resolution process $\langle \mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(n)} \rangle$ it even holds that $Conflicts(\mathcal{P}^{(i+1)}) \subsetneq Conflicts(\mathcal{P}^{(i)})$ for $1 \leq i < n$. Thus, (P1) holds. As every conflict resolution step consists of solely replacing a conflicting rule with its λ-extension of the form (10), (P2) holds. By Proposition 4, property (P3) holds also. Therefore, a conflict resolution process using conflict-resolving λ-extensions leads to a uniformly non-contradictory program core in a minimally invasive manner. Including a domain-specific expert into the selection of the most suitable λ-extensions can furthermore ensure that the resulting non-contradictory program core maintains its professionally adequate knowledge base.

3.6 Missing λ-extension

Under some circumstances, no λ-extensions can be generated for a rule $r \in \mathcal{P}$. In these cases a blanket for $Adv(r)$ cannot be found. Determining whether a blanket for $Adv(r)$ exists can become quite cumbersome and complex since the relationships of atom-related literals between all adversarial rules have to be examined. A small fraction of such possible literal interactions that can make the finding of a blanket impossible will be illustrated in the following example.

Example 15
Suppose the following program $\mathcal{P}_{15}$:

$$r_1\colon x \leftarrow a, d. \qquad r_2\colon \overline{x} \leftarrow a, b. \qquad r_3\colon \overline{x} \leftarrow \overline{c}. \qquad r_4\colon \overline{x} \leftarrow c, {\sim}b.$$

Program $\mathcal{P}_{15}$ has conflicts $(r_1, r_2), (r_1, r_3), (r_1, r_4)$ and there does not exist a blanket for r_1. This can be easily shown by looking at each potential candidate literal: Literal a is already in $B(r_1)$ and can therefore not be in a blanket. Since atom b (resp. c) appears in $B(Conflicts(r_1))$ as default (resp. classically) complementary literals, viz. r_2, r_4 (resp. r_3, r_4), b (resp. c) cannot be part of a blanket either.

There are several workarounds that allow the resolution of such conflicts nevertheless. One possible way to resolve such conflicts can be the *partitioning* of conflicts, that is, given a rule r with several adversaries $Adv(r)$, one could first determine a subset $Adv' \subset Adv(r)$ for which a blanket can be computed. This procedure can be repeated until all conflicts of r' are resolved. The following example illustrates how partitioning can be used to obtain a solution if a specific rule of the program (in this case r_1) should primarily be modified.

Example 16 (*Example 15 contd.*)
Suppose program $\mathcal{P}_{15}$ from Example 15. First, conflicts (r_1, r_2) and (r_1, r_3) can be resolved simultaneously by generating λ-extensions from the blankets for $Adv' = \{r_2, r_3\}$. Two of the resulting extensions are $\lambda_1 = \{{\sim}b, {\sim}\overline{c}\}$ and $\lambda_2 = \{{\sim}b, c\}$. Applying λ_1 to $\mathcal{P}_{15}$ results in program $\mathcal{P}'_{15}$:

$$r'_1\colon x \leftarrow a, d, {\sim}b, {\sim}\overline{c}. \qquad r_2\colon \overline{x} \leftarrow a, b. \qquad r_3\colon \overline{x} \leftarrow \overline{c}. \qquad r_4\colon \overline{x} \leftarrow c, {\sim}b.$$

Then, the remaining conflict (r'_1, r_4) can be resolved by applying the unique solution $\lambda_3 = \{{\sim}c\}$ which in turn results in program $\mathcal{P}''_{15}$:

$$r''_1\colon x \leftarrow a, d, {\sim}b, {\sim}\overline{c}, {\sim}c. \qquad r_2\colon \overline{x} \leftarrow a, b. \qquad r_3\colon \overline{x} \leftarrow \overline{c}. \qquad r_4\colon \overline{x} \leftarrow c, {\sim}b.$$

Note that the reason that a solution like $\lambda = \{{\sim}b, {\sim}c, {\sim}\overline{c}\}$ cannot be obtained with the general conflict resolution approach is because of the restriction that blankets cannot contain multiple literals of the same atom even though reconcilable literals do not interfer with the applicability of a rule (as can be seen in Example 15). But in order to keep the resolution approach simple and pragmatic, such a restriction on blankets was deemed necessary.

However, whenever conflicts are separately resolved, the final outcome can very much depend on the actual partitions, the order in which the different conflicts are resolved, and the choice of the respective λ-extensions along the way.

Example 17 (*Example 16 contd.*)

Reconsider the results in Example 16. Note that (r'_1, r_4) in $\mathcal{P}'_{15}$ only has the unique solution $\lambda_3 = \{\sim c\}$ because for the resolution of the previous conflicts λ_1 was chosen. Applying λ_2 to $\mathcal{P}_{15}$ of Example 15 instead of λ_1 would result in program $\mathcal{P}'''_{15}$:

$$r'_1 \colon x \leftarrow a, d, c, \sim b. \qquad r_2 \colon \overline{x} \leftarrow a, b. \qquad r_3 \colon \overline{x} \leftarrow \overline{c}. \qquad r_4 \colon \overline{x} \leftarrow c, \sim b.$$

It is easy to see that now for conflict (r_1, r_4), there exist no possible λ-extensions anymore as $B(r_4) \subset B(r_1)$.

Another possible workaround could be a preceding resolution of conflicts of some adversarial rules $r' \in Adv(r)$, that is, if a blanket for $Adv(r)$ cannot be found, one could resolve the conflicts of some $r' \in Adv(r)$ first and then resolve the remaining conflicts of r. This workaround is illustrated in the following example:

Example 18 (*Example 15 contd.*)

Again, suppose program $\mathcal{P}_{15}$ from Example 15. Since r_1 contains literal d that does not appear in $B(Conflicts(r_1))$, instead of resolving the conflicts of r_1, the conflicts of the adversarial rules could be resolved consecutively, in this case even with the same λ-extension, namely conflicts $(r_2, r_1), (r_3, r_1), (r_4, r_1)$ with either λ-extension $\lambda_3 = \{\sim d\}$ or $\lambda_4 = \{\overline{d}\}$. Therefore, one possible resolved program $\mathcal{P}''''_{15}$ would consist of the following rules:

$$r_1 \colon x \leftarrow a, d. \qquad r'_2 \colon \overline{x} \leftarrow a, b, \sim d \qquad r'_3 \colon \overline{x} \leftarrow \overline{c}, \sim d. \qquad r'_4 \colon \overline{x} \leftarrow c, \sim b, \sim d.$$

Examples 16 and 18 show that if the resolution of conflicts of a rule cannot be carried out in one go, there are possible workarounds. But, as shown with Example 17, even if the conflict resolution is partitioned into multiple ones, the choice of a solution can influence the success of remaining conflict resolution attempts. Thus, in such individual cases, both, the order in which the conflicts are solved and the choice of the solutions have to be executed with care. Apart from partitioning conflicts and modifying adversarial rules, there is always the possibility to introduce new atoms to the bodies of a conflict, for example, adding a new atom A to $B(r)$ and $\overline{A}$ to the bodies $B(r')$ of all adversaries. This, however, would violate our requirement of extensions being informative.

Improving our strategies with respect to such deficiencies right from the beginning, for example, finding a most suitable order in which conflicts are resolved, is part of our ongoing work. Some enhancements of our approach are considered in the next section.

4 Enhancements

In the following, we will outline possible enhancements to the presented resolution approach. Section 4.1 outlines how the multiple resolution processes can be used to resolve many-to-many conflicts and in Section 4.2, we show how the current resolution approach can also be used to deal with inconsistency-causing constraints. In the last two subsections, we will show methods to compute additional information for the knowledge expert to find the most suitable solutions efficiently. In Section 4.3, we will present a method to sort the detected conflicts in a program by the impact the conflicting rules have on other rules. We extend this method in Section 4.4 to introduce the concept of λ-*scores* that enable the ordering of λ-extensions.

4.1 Many-to-many conflicts

So far, the presented approach handles *1-to-many* conflicts, that is, the presented approach modifies a single rule r such that the conflicts with n different rules $r' \in Adv(r)$ are resolved simultaneously. It is conceivable that instead of a single rule r, a program $\mathcal{P}$ can contain a set $\mathcal{R}$ of m different rules r with $H(r) = L(m > 1)$ which results in considering what we will call *many-to-many* conflicts. Currently, in those cases the resolution process have to be applied on each such $r \in \mathcal{R}$ separately and consecutively. Depending on the individual case, it can also be reasonable to modify a rule $r' \in Adv(r)$.

Example 19
Consider the following ELP:

$$\mathcal{P}_{19}: \quad r_1: x \leftarrow a, e. \qquad r_2: x \leftarrow b, e. \qquad r_3: \overline{x} \leftarrow f. \qquad r_4: \overline{x} \leftarrow g.$$

In $\mathcal{P}_{19}$, there are conflicts between $\{r_1, r_2\}$ and $\{r_3, r_4\}$. One way to resolve all conflicts is to extend r_1 and r_2 by a λ-extension, respectively. Since the bodies of r_3 and r_4 do not contain atom-related literals, each λ-extension will contain two literals. If, on the other hand, the knowledge expert decides to modify r_3 and r_4 instead, a possible λ-extension for both rules can consist of a single literal, viz. either $\sim e$ or $\overline{e}$.

Example 19 shows that the order in a conflict pair (r, r') can decide over the "complexity" of its resolution.

4.2 Conflict resolution with constraints

In general, (integrity) constraints in answer set programs serve to weed out unwanted answer sets. One way to implement such a functionality is to map a constraint like $\leftarrow a, b.$ to a rule $z \leftarrow a, b, \sim z.$ where z is a newly introduced atom (Gebser *et al.* 2012). Even though the derivation of z due to such constraints technically causes incoherence and not contradiction, we will cover the handling of a specific type of constraints r_c as we can show that the presented conflict resolution approach can be used to prevent the advent of incoherence due to such rules r_c. In the following, we will examine constraints of the form

$$r_c: \leftarrow K, L, \tag{12}$$

where K, L are internal atoms.

Recall that in logic programs, classical negation is actually syntactic sugar: Given the normal logic program $\mathcal{P}^+$ of $\mathcal{P}$, classical negation can be incorporated by adding a constraint $\leftarrow A, A'.$ for every atom $A \in \mathcal{A}$ (Gelfond and Lifschitz 1991).

Example 20
Suppose the following program $\mathcal{P}_{20}$:

$$r_1: x \leftarrow a. \qquad\qquad\qquad r_2: \overline{x} \leftarrow b.$$

This program is actually interpreted as a positive program $\mathcal{P}_{20}^+$:

$$r_1: x \leftarrow a. \qquad r_2': x' \leftarrow b. \qquad r_x: \leftarrow x, x'.$$

Example 20 shows that resolving the conflict between (r_1, r_2') also ensures that r_x is satisfied. As a generalization of this idea, we can therefore consider constraints as implicit

pointers to potential conflicts. Consequently, the satisfaction of a constraint of form (12) can be ensured by treating every pair of rules r, r' in $\mathcal{P}$ with $H(r) = L, H(r') = K$ as *implicit conflicts* (r, r') and resolving them using the method for the resolution of conflicts via λ-extensions.

Example 21
Suppose the following program $\mathcal{P}_{21}$ with internal atoms a, d and external atoms b, c, e, f:

$$r_1: a \leftarrow b, \sim c. \qquad r_2: a \leftarrow b, \sim f. \qquad r_3: d \leftarrow e. \qquad r_4: \leftarrow a, d.$$

Then, (r_3, r_1) and (r_3, r_2) can be interpreted as conflicts. To resolve these two conflicts simultaneously, $\{\sim b\}$ can be added as a λ-extension to r_3. The modified program $\mathcal{P}'_{21}$ then has the following rules:

$$r_1: a \leftarrow b, \sim c. \qquad r_2: a \leftarrow b, \sim f. \qquad r'_3: d \leftarrow e, \sim b. \qquad r_4: \leftarrow a, d.$$

It is easy to see that a constraint of form (12) can be omitted once the implicit conflicts of the constraint are resolved with fitting λ-extensions. Thus, rule r_4 is redundant in program $\mathcal{P}'_{21}$, but one has to keep in mind that future, more comprehensive updates can lead to answer sets that contain literals a and d.

4.3 Conflict order

As stated at the beginning, the presented approach is a basic method that can be extended in many respects depending on the requirements of the user. A useful functionality regarding the workflow of conflict resolution is the order in which the conflicts are presented to the expert. It is important that the knowledge expert understands how the order of the conflicts is generated as any enhancement in the proposed framework must not negatively influence the actual decision regarding a resolution but rather aid the expert in finding the most suitable solution efficiently. One easily comprehensible implementation utilizes the *affected rule count* (inspired by Abdelhalim *et al.* 2009) of the conflicting literals that reflects how many rules are potentially affected by a specific rule. For that, we first define the set $\mathcal{R}$ of all rules that are *affected by a rule* r_1 and compute the rule impact of r_1 by counting its affected rules.

Definition 10 (Affected Rules and Affected Rule Count)
The set $\mathcal{R}_{r_1}$ of *rules affected by a rule* $r_1 \in \mathcal{P}$ is defined as

$$\mathcal{R}_{r_1} = \{r \in \mathcal{P} \mid atom(H(r_1)) \in Atom(B(r)), (B(r_1) \cup B(r)) \text{ is consistent}\}.$$

The *affected rule count* $AR(r_1)$ of r_1 is then defined by the number of affected rules, viz.

$$AR(r_1) = |\mathcal{R}_{r_1}|.$$

Example 22 (Example 4 continued)
Let $\mathcal{P}_{22}$ be program $\mathcal{P}_4$ extended by the following rules:

$$r_5: drugB \leftarrow eligX, condAAdv, \overline{drugE}.$$
$$r_6: drugC \leftarrow eligX, \sim condAAdv, highLCount.$$
$$r_7: drugD \leftarrow \overline{eligX}, condAAdv, \overline{preTreatedN}.$$

In $\mathcal{P}_{22}$, rules r_5 and r_7 are affected by r_1. Rule r_6 is not affected as $B(r_1)$ and $B(r_6)$ are not consistent due to atom *condAAdv*, that is, literal *condAAdv* occurs in $B^+(r_1)$ and in $B^-(r_6)$. Ergo for r_1, we yield the affected rule count $AR(r_1) = 2$.

The affected rule count can provide the expert with a first impression (or possibly confirm own conjectures) of the possible impact that the given conflict and consequently the changes made to one of the conflicting rules have within the program. It can contribute additional information to the expert to anticipate the effect of a possible solution. Hence when using this order-criterion, it is on the expert to decide which conflict they want to solve first, for example, they can handle those conflicts with the least impact on other rules first. Additionally, such computed values can also help to decide which rules of a conflict to modify, for example, given a rule $r \in \mathcal{P}$, it can be beneficial to modify one rule in *conflicts*(r) first and then modify r in order to resolve all remaining conflicts of r (see Section 4.1).

4.4 λ-scores

In the previous section, we presented a method that aims to improve the user's first major task during the resolution process which is choosing a conflict out of all detected conflicts. In the next step, the user has to decide on a solution for a picked conflict. To facilitate the finding of a most suitable solution for a chosen conflict (r, r'), an ordering of the computed suggestions can be established. For that one can assign a score to each λ-extension that is calculated using different criteria. In this section, we will outline one prototypical enhancement that adds a numerical value to each generated λ-extension named *λ-score*. We will present a basic approach to compute λ-scores by utilizing the previously introduced affected rule count to quantify the *effect* a rule modification can have. Suppose a conflict (r_1, r_2) and a solution r_1' which can either be a computed solution (λ-extended rule) or a modified custom solution, respectively, for this conflict. In order to reflect the effect this solution can have on the overall program, one can look at the size of the difference between the affected rules of r_1 and the λ-extended rule r_1' which we will call the *effect of the modification of r_1 to r_1'*.

Definition 11 (*Effect*)
Let (r_1, r_2) be a conflict in $\mathcal{P}$ and r_1' a generated, λ-extended rule of r_1. Let furthermore $\mathcal{R}_{r_1}$ and $\mathcal{R}_{r_1'}$ be the respective sets of affected rules. Then, the *effect Eff*(r_1, r_1') of this modification is defined by

$$\textit{Eff}(r_1, r_1') = \mathcal{R}_{r_1} - \mathcal{R}_{r_1'}.$$

Note that as λ-extended rules always expand the body of the original rule and therefore are more specific, any generated solution will affect the same rules or a subset of the rules that are affected by the original rule, that is, $\mathcal{R}_{r_1'} \subseteq \mathcal{R}_{r_1}$ holds. The higher the effect of a solution, the more rules are not affected anymore after the modification w.r.t. the rules affected by the original rule. The effect of a suggested modification can therefore be used as the λ-score of the suggestion. If for example an expert wants to examine the most cautious solutions, the solutions with the least effect can be presented first.

Example 23 (Example 10 continued)

Suppose, the expert wants to resolve the conflict in $\mathcal{P}_{22}$ by changing r_1 to r'_1 with

$$r'_1 \colon eligX \leftarrow \overline{highLCount}, preTreatedN, ctrIndR.$$

As rule r_5 is the only rule affected by r'_1, we get the effect

$$Eff(r_1, r'_1) = \{r_5, r_7\} - \{r_5\} = 1.$$

The effect of a modification tells the expert how impactful a change is regarding the affected rules, that is, how many additional and less rules are affected after the changes. The interpretation of the effect of modifications that is offered at this point can be seen as a first proposal to produce the λ-scores. The corresponding method for their computation can thus serve as groundwork where individual enhancements and modifications are possible. The presented definition of modification effects is an illustrative example to show how λ-extensions can hypothetically be ordered. For a given conflict, the user can inspect the suggestions sorted by their effect, starting with those that have the least effect. This way, they can look for and prioritize solutions that have the least impact on the program.

In analogue to the previously described method for ordering conflicts, any implementation for the computation of λ-scores must not negatively influence the expert regarding their decisions, that is, enhancements like ordering conflicts and suggestions should not incentivize the expert to choose solutions with the sole intention to resolve all conflicts in the technically easiest and quickest way possible without considering the underlying errors and problems that led to the conflict, nor should it bias the expert's thought process in any other negative way. In other words, the goal of such enhancements is to improve the finding of solutions that reflect the actual professional expertise best and simultaneously provide robustness for future updates (i.e. mitigate the potential for future conflicts), while not compromising any decision on a solution during the process.

Establishing a score for each suggestion allows to generate an order over all suggestions which in turn can be used to show the knowledge expert those suggestions first that are more relevant according to their criteria. If a *semi-automated* approach is wanted, the knowledge expert could also define a specific score threshold so that suggestions can be applied without the expert's involvement. Then, if there exist suggestions λ with a score below such a threshold, the conflict can be resolved directly by using every λ with the lowest score as a solution. Enhancements like these that gather and use additional *metadata* can help to add a highly customizable abstraction layer between the knowledge expert and the actual knowledge base where the expert can input information about the program elements in order to resolve a conflict without the need for any technical knowledge about answer set programming while ensuring that the resulting rules are still representing correct knowledge in the professional sense.

5 Conclusion and future work

In this paper, we proposed a method to modify answer set programs such that they remain consistent given any (allowed) instance data. We provided an analysis how contradictions in a program can arise and discussed the necessary requirements rules have to satisfy in order to prevent contradictions while keeping the degree of change to a minimum. Based

on that, we arrived at the notion of λ-extensions and we presented a strategy that gathers all conflict-resolving λ-extensions for a conflicting rule that comprise the atoms occurring in the conflicting rules. Extending a rule by a corresponding conflict-resolving λ-extension resolves all conflicts of this rule simultaneously. As a consequence, a conflict resolution process that uses the proposed strategy for computing conflict-resolving λ-extensions eventually yields a uniformly non-contradictory program core.

The presented approach constitutes crucial groundwork for the implementation of a larger (application) framework for updating and maintaining answer set programs. It enables the knowledge expert to inspect contradictory statements in a program and update rules interactively in view of new information given by an update. The suggestions that are generated by this algorithm can be viewed as a baseline for modifications. As mentioned in Example 10, applying a suggested λ-extension to a rule r can lead to a "stricter" λ-extended rule r'. Whether this makes sense or not, is highly dependent on the professional meaning of these rules. Our approach allows the knowledge expert to decide whether a suggested enhancement should be applied as is or additional (manual) modifications should follow the applied suggestion. But as shown with Proposition 5, every λ-extension satisfies the minimal requirement of resolving the corresponding conflicts simultaneously and can therefore used as a reliable starting point.

This paper adresses one part of an interactive solution for conflict resolution in logic programs as proposed in Thevapalan and Kern-Isberner (2020). The running example illustrates that the proposed framework can pose a highly beneficial tool in the medical sector and other domains. We argue that any area where experts face complex decisions based on knowledge that is not solely comprised of facts but also on practical knowledge (experience) and continually growing and changing (adapted) expertise can highly profit from such a framework to efficiently build up and maintain an adequate knowledge base.

The ideas and results discussed in this paper can be adapted and extended in various ways. In Thevapalan *et al.* (2022), the authors show how conflicts can be resolved in cases where the actual conflicting rules shall remain unchanged. For that, the concept of λ-extensions is picked up to generate solutions that restore consistency. Aside from contradictions, extended logic program can also become inconsistent due to incoherence. A method to pinpoint the causes of incoherence in logic programs and strategies to reestablish coherence in a program are developed in Thevapalan *et al.* (2021).

Since in the presented conflict resolution strategy, the body of one of the conflicting rules is extended, one possible extension of the approach is the compilation of strategies for the remaining types of program modifications, namely reduction of rule bodies, replacing body literals, removing rules as a whole and adding new rules.

As outlined Section 4, we see numerous ways to extend the presented approach. In the future, we want to adapt the method of conflict resolution to resolve many-to-many conflicts in a more convenient way that generates λ-extensions based on all involved rules that at best does not require the repeated use of a resolution.

Additionally, we want to work out methods to interactively compute λ-scores (Section 4.4) in order to find suitable solutions more efficiently.

In each conflict resolution step during a conflict resolution process, solutions for a conflict are computed independently of other conflicts. In order to contemplate dependencies between conflicts, future work will also include the development of suitable heuristics to determine which rules of a conflict to modify (since a conflict-relation is symmetrical) and

refine the order in which the different conflicts should be resolved. Besides taking into account the connections between conflicts, such heuristics could also consider individual preferences and conditions given by the expert, for example, a set of rules that shall not be modified, rule weights etc.

The other major functionality that is part of our ongoing work is the actual interaction process between the expert and the framework. These interactions must tackle two problems: explaining the cause of the respective conflict to the user and choosing the most suitable solution for it. Instead of burden the user with all available information, we suggest a dialogical approach where the user can reach an informed decision after a back-and-forth dialogue with the framework. Using argumentative dialogue theories (Caminada 2017; Modgil and Caminada 2009; Walton and Krabbe 1995) could enable the framework to "communicate" with the user where they can ascertain the underlying causes of the conflict and successively attain the most suitable suggestion.

References

ABDELHALIM, A., TRAORÉ, I. AND SAYED, B. 2009. RBDT-1: A new rule-based decision tree generation technique. In *Rule Interchange and Applications, International Symposium, RuleML 2009, Las Vegas, Nevada, USA, 5–7 November 2009. Proceedings*, G. Governatori, J. Hall and A. Paschke, Eds. Lecture Notes in Computer Science, vol. 5858. Springer, 108–121.

ALFERES, J. J., LEITE, J. A., PEREIRA, L. M., PRZYMUSINSKA, H. AND PRZYMUSINSKI, T. C. 1998. Dynamic logic programming. In *1998 Joint Conference on Declarative Programming, APPIA-GULP-PRODE'98, A Coruña, Spain, 20–23 July 1998*, J. L. Freire-Nistal, M. Falaschi and M. V. Ferro, Eds., 393–408.

ALFERES, J. J., LEITE, J. A., PEREIRA, L. M., PRZYMUSINSKA, H. AND PRZYMUSINSKI, T. C. 2000. Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming 45,* 1-3, 43–70.

BERGE, C. 1989. *Hypergraphs - Combinatorics of Finite Sets.* North-Holland Mathematical Library, vol. 45. North-Holland.

CAMINADA, M. 2006. Well-founded semantics for semi-normal extended logic programs. In *Proc. 11th Int'l Workshop on Nonmonotonic Reasoning*, 103–108.

CAMINADA, M. 2017. Argumentation semantics as formal discussion. *Journal of Applied Logics 4,* 8, 2457–2492.

CAMINADA, M. AND SAKAMA, C. 2006. On the existence of answer sets in normal extended logic programs. In *ECAI 2006, 17th European Conference on Artificial Intelligence, 29 August– 1 September 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, G. Brewka, S. Coradeschi, A. Perini and P. Traverso, Eds. Frontiers in Artificial Intelligence and Applications, vol. 141. IOS Press, 743–744.

COSTANTINI, S. 2006. On the existence of stable models of non-stratified logic programs. *Theory and Practice of Logic Programming 6,* 1-2, 169–212.

EITER, T., FINK, M., SABBATINI, G. AND TOMPITS, H. 2002. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming 2,* 6, 711–767.

GEBSER, M., KAMINSKI, R., KAUFMANN, B. AND SCHAUB, T. 2012. *Answer Set Solving in Practice.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9,* 3/4, 365–386.

GHOSH, A. 2004. Understanding medical uncertainty: A primer for physicians. *Journal of Association of Physicians of India 52,* 739–742.

INOUE, K. 1993. *Studies on Abductive and Nonmonotonic Reasoning*. Ph.D. thesis, Kyoto University.

MODGIL, S. AND CAMINADA, M. 2009. Proof theories and algorithms for abstract argumentation frameworks. In *Argumentation in Artificial Intelligence*. Springer, 105–129.

PRZYMUSINSKI, T. C. 1991. Three-valued nonmonotonic formalisms and semantics of logic programs. *Artificial Intelligence 49,* 1-3, 309–343.

REITER, R. 1980. A logic for default reasoning. *Artificial Intelligence 13,* 1-2, 81–132.

SCHULZ, C. 2017. *Developments in Abstract and Assumption-based Argumentation and their Application in Logic Programming*. Ph.D. thesis, Imperial College London, UK.

SCHULZ, C., SATOH, K. AND TONI, F. 2015. Characterising and explaining inconsistency in logic programs. In *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, 27–30 September 2015. Proceedings*, F. Calimeri, G. Ianni and M. Truszczynski, Eds. Lecture Notes in Computer Science, vol. 9345. Springer, 467–479.

THEVAPALAN, A., HAUPT, K. AND KERN-ISBERNER, G. 2022. Towards causality-based conflict resolution in answer set programs. In *16th International Conference on Logic Programming and Non-monotonic Reasoning, LPNMR 2022, Genova Nervi, Italy, 5–9 September 2022. Proceedings*. Lecture Notes in Computer Science, vol. 13416. Springer. to be published.

THEVAPALAN, A., HEYNINCK, J. AND KERN-ISBERNER, G. 2021. Establish coherence in logic programs modelling expert knowledge via argumentation. In *Proceedings of the International Conference on Logic Programming 2021 Workshops co-located with the 37th International Conference on Logic Programming (ICLP 2021), Porto, Portugal (virtual), 20–21 September 2021*, J. Arias, F. A. D'Asaro, A. Dyoub, G. Gupta, M. Hecher, E. LeBlanc, R. Peñaloza, E. Salazar, A. Saptawijaya, F. Weitkämper and J. Zangari, Eds. CEUR Workshop Proceedings, vol. 2970. CEUR-WS.org.

THEVAPALAN, A. AND KERN-ISBERNER, G. 2020. Towards interactive conflict resolution in asp programs. In *18th International Workshop on Non-Monotonic Reasoning, Workshop Notes*, M. V. Martínez and I. Varzinczak, Eds., 29–36.

VAN GELDER, A., ROSS, K. A. AND SCHLIPF, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38,* 3, 620–650.

WALTON, D. AND KRABBE, E. C. W. 1995. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of New York Press, Albany, NY, USA.