Resolving Constraint-Induced Inconsistencies in ASP Using lambda-Extensions

Andre Thevapalan[1,*], Gabriele Kern-Isberner[1]

[1]TU Dortmund, Otto-Hahn-Straße 12, 44227 Dortmund, Germany

Abstract

Designing reliable logic programs becomes especially challenging when constraints are involved, i.e., rules that must never be violated, no matter the input. Therefore, constraints may induce conflicts with other rules that derive the literals in their bodies, causing constraint-induced inconsistencies (CI-inconsistencies). In this paper, we introduce a new consistency notion for answer set programming: robust CI-consistency, which ensures that a program's constraints are respected under all admissible sets of facts. Our focus lies on positive constraints, a natural and intuitive class that avoids default negation and expresses hard requirements on the program's output. To support knowledge engineers in early stages of modelling, we present a method that allows the expert to define general constraints and selectively refines only the rules that could lead to violations. This is achieved by extending the idea of λ-extensions, originally developed for handling contradictions caused by strong negation. By systematically identifying problematic rule combinations and adjusting them just enough to prevent conflicts caused by future instance data, our approach yields encodings that are correct by design in any case–without deleting rules or depending on specific data. This framework helps bridge the gap between early, intuitive modelling and robust, instance-independent correctness.

Keywords

Answer Set Programming, Inconsistency, Constraints, lambda-Extensions, Conflicts, Program Repairs

1. Introduction

Answer Set Programming (ASP) is a declarative paradigm for knowledge representation and reasoning, well-suited to solving combinatorial problems. In practice, an ASP program is usually divided into two components: the encoding, which defines the general rules of the problem, and the instance, which provides concrete input data as facts. This separation enables a single encoding to be applied to multiple instances, akin to executing software with different input data.

However, in high-stakes applications such as medical decision support systems, it is critical to ensure that an encoding behaves reliably across all admissible instances (e.g., patient data). Constraints can play a vital role in such contexts–for example, by preventing the recommendation of therapies or drugs with known adverse interactions. Domain experts may define hard constraints to enforce compliance with clinical guidelines or to block recommendations involving harmful combinations. While such constraints are essential for ensuring correctness, they can inadvertently introduce inconsistencies if the encoding fails to anticipate rare or exceptional cases. This, in turn, may cause the program to become inconsistent, rendering the encoding unusable for certain instances.

Prior works [1, 2] have addressed the challenge of maintaining robust consistency of encodings, i.e., ensuring that the combination of encoding and instance does not lead to inconsistency when the source of inconsistency are strongly complementary literals. In that context, a method was introduced to construct encodings that remain consistent under all admissible instances, effectively shielding the program from inconsistency due to such literals and ensuring its applicability in all contexts.

In this work, we extend this approach to constraint-induced inconsistencies (*CI-inconsistencies*). We enable knowledge engineers to define problem encodings in a more general way, without needing to account for every exception or rare case individually in the first place. The system will check if any

23rd International Workshop on Nonmonotonic Reasoning, November 11–13, 2025, Melbourne Australia

*Corresponding author.

andre.thevapalan@tu-dortmund.de (A. Thevapalan); gabriele.kern-isberner@cs.tu-dortmund.de (G. Kern-Isberner)

0000-0001-5679-6931 (A. Thevapalan); https://orcid.org/0000-0001-8689-5391 (G. Kern-Isberner)

admissible instances lead to the activation of constraints. If such instances arise, the system signals the issue, allowing the engineer to fine-tune the relevant rules in the encoding. This ability to dynamically refine encodings while maintaining the integrity of the constraints makes the approach particularly well-suited for complex, real-world scenarios, ensuring robust, consistent, and tailored solutions. Our approach motivates the usage of constraints by emphasising their critical role in ensuring that the program behaves as intended. Instead of manually handling each potential inconsistency, engineers can specify constraints that act as strict guards, preventing undesired answer sets. This allows them to focus on creating flexible, reusable encodings, while leaving the responsibility of ensuring correctness to the constraints, which guide the system in preventing unintended behaviours.

The remainder of this paper is structured as follows. Section 2 introduces the necessary preliminaries on answer set programming to establish the formal background. In Section 3, we review related work and summarise the existing approach in [1], which addresses inconsistency arising from strongly complementary literals through conflict-resolving λ-extensions. Section 4 presents our novel method for achieving robust CI-consistency in problem encodings by systematically resolving constraint-induced inconsistencies using a refined extension of the λ-framework. Finally, Section 5 concludes the paper and outlines directions for future research.

2. Preliminaries

In this paper, we examine the propositional fragment of non-disjunctive *extended logic programs* (ELPs) [3], which are finite sets of rules defined over a set of atoms $\mathcal{A}$. A literal l is either an atom $a \in \mathcal{A}$ or its strong negation $\neg a$. For a literal l, the *strongly complementary* literal $\overline{l}$ is defined to be $\neg a$ if $l = a$, and a otherwise. The set $\mathcal{L}_\mathcal{A} = \mathcal{A} \cup \{\overline{a} \mid a \in \mathcal{A}\}$ denotes the set of all literals based on $\mathcal{A}$.

A *default-negated literal* l, also referred to as a *default literal*, is written as $\sim l$. An *extended literal* $l^\star$ is either a literal l or its default negation $\sim l$. We also extend $\sim$ to sets: for a set S of literals, $\sim S = \{\sim l \mid l \in S\}$. We will use $\sim$ as a unary junctor in order to describe the *default complement* of an extended literal: for an extended literal $l^\star$, the *default-complementary* literal $\sim l^\star$ is defined as $\sim l$ if $l^\star = l$, and l otherwise.

A set of extended literals is said to be *inconsistent* if it contains strongly or default-complementary literals. Two extended literals $l^\star$ and $k^\star$ are referred to as *mutually exclusive* if they are strongly complementary or default-complementary.

For each extended literal $l^\star$, we write $atom(l^\star)$ to denote the atom on which $l^\star$ is based. The underlying set of atoms in a set X of extended literals is denoted by $Atom(X) = \{atom(l^\star) \mid l^\star \in X\}$.

A rule r is of the form:

$$H(r) \leftarrow B^+(r), \sim B^-(r). \tag{1}$$

where $H(r)$, called the *head of* r, is either a singleton set $\{l\}$ for some literal l, or the empty set $\emptyset$. The sets $B^+(r)$ and $B^-(r)$, called the *positive* and *negative body of* r, respectively, are (possibly empty) sets of literals. The *body of* r is defined as $B(r) = B^+(r) \cup \sim B^-(r)$.

A rule r is called a *fact* if $B(r) = \emptyset$, and a *constraint* if $H(r) = \emptyset$. A constraint r is said to be *positive* if $B^-(r) = \emptyset$; note that positive constraints may still contain strong negation.

An *extended logic program (ELP)* is a set of rules of the form (1). In the remainder of this paper, we assume that any ELP $\mathcal{P}$ is over the set $\mathcal{A}$ of atoms occurring in $\mathcal{P}$.

Given an ELP $\mathcal{P}$, we distinguish the following components: the set $\Phi_\mathcal{P}$ of facts, referred to as the *problem instance of* $\mathcal{P}$; the set $\Psi_\mathcal{P}$ of constraints; and set $\Pi_\mathcal{P}$ of all remaining rules, referred to as the *problem encoding of* $\mathcal{P}$. To each problem encoding Π, we associate its set of *external atoms*, denoted by $\mathcal{E}(\Pi) \subseteq \mathcal{A}$, defined as the set of atoms that occur in Π but not in the head of any rule in Π.

We illustrate the components of an ELP using the following running example.

Example 1. *Consider a simplified medical decision-support system that recommends treatments based on suspected diagnoses, which we use as a running example.*

$$r_1\colon leuk \leftarrow anem, \sim abnSk. \qquad\qquad r_2\colon mela \leftarrow lesn, \sim swLn.$$

r_3: $chem \leftarrow leuk, \overline{imsp}.$ r_4: $imtx \leftarrow mela, \sim aid.$

r_5: $\leftarrow chem, imtx.$

This program models the diagnosis and treatment process for leukaemia ($leuk$) *and* melanoma ($mela$)*, two cancers that typically do not occur together except in rare cases. If a patient has* anaemia ($anem$) *and no* visible skin abnormalities ($abnSk$)*, leukaemia is suspected. If the patient has a* lesion ($lesn$) *but no* swollen lymph nodes ($swLn$)*, melanoma is diagnosed. If leukaemia is confirmed and the patient is not* immunosuppressed ($imsp$)*,* chemotherapy ($chem$) *is prescribed. If melanoma is diagnosed and there is no* autoimmune disorder (aid)*,* immunotherapy ($imtx$) *is recommended.*

The integrity constraint in r_5 ensures that both therapies are not prescribed together. This is because chemotherapy and immunotherapy target different mechanisms and may have adverse interactions if used in combination, making their joint application inappropriate for the patient's safety.

The rules r_1–r_4 form the problem encoding Π_1, while $\Psi_1 = \{r_5\}$ defines the constraint set, preventing the simultaneous prescription of both therapies. Possible problem instances Φ_1 are facts corresponding to subsets of the external atoms $\mathcal{E}(\Pi_1) = \{anem, abnSk, lesn, swLn, imsp, aid\}$.

We now move on to the semantics of ELPs.

Let S be a set of literals. A literal l is said to be *true in* S if $l \in S$, *false in* S if $\bar{l} \in S$, and *unknown* otherwise. A rule is *applicable under* S if $B^+(r) \subseteq S$ and $B^-(r) \cap S = \emptyset$. The set S *satisfies the body* $B(r)$ of a rule r if r is applicable under S. The set S *satisfies a rule* r if and only if $H(r) \cap S \neq \emptyset$ whenever S satisfies $B(r)$. A set S of literals *supports a rule* r if r is applicable and satisfied under S. We say r is *unsatisfiable* if there exists no consistent set S of literals s.t. r is applicable under S.

Note: If r is a constraint, then S satisfies r if S does not satisfy $B(r)$. Conversely, if S satisfies the body of a constraint, we say that S *violates* r.

In general, an *answer set of an ELP* $\mathcal{P}$ is determined by its reduct. The *reduct* $\mathcal{P}^S$ of a program $\mathcal{P}$ relative to a set S of literals is defined as:

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

Note that any reduct contains no default negation.

For programs $\mathcal{P}$ without default negation, we define the *closure* $Cl(\mathcal{P})$ of $\mathcal{P}$ as the smallest subset $S \subseteq \mathcal{L}_\mathcal{A}$ such that S satisfies all rules in $\mathcal{P}$. Notably, similar to Horn logic programs, each ELP without default negation has exactly one closure $Cl(\mathcal{P})$, which may, however, be inconsistent (cf. [4]).

Let $\mathcal{P}$ be an ELP over $\mathcal{A}$, and assume $\mathcal{P}$ contains no default negation. The *answer set of* $\mathcal{P}$ is defined as follows:

(a) if $Cl(\mathcal{P})$ is consistent, then $Cl(\mathcal{P})$ is the unique answer set of $\mathcal{P}$;

(b) otherwise, the answer set of $\mathcal{P}$ is defined to be $\mathcal{L}_\mathcal{A}$, i.e., the complete set of literals over $\mathcal{A}$.

A set S of literals is an *answer set of an ELP* if it is the answer set of the reduct $\mathcal{P}^S$ [3]. Specifically, a consistent set S is an answer set of $\mathcal{P}$ if:

$$S = Cl(\mathcal{P}^S). \tag{2}$$

We denote the set of all answer sets of $\mathcal{P}$ by $AS(\mathcal{P})$.

Example 2 (Example 1 contd.)**.** *Let $\mathcal{P}_2$ be a program consisting of problem encoding Π_1 and constraints Ψ_1 from Example 1, and the following problem instance $\Phi_2 = \{anem.\}$. Then, $\mathcal{P}_2$ has the unique answer set $S_1 = \{anem, leuk, chem\}$.*

3. Previous and Related Work

The authors in [5] propose a stepwise debugger for extended logic programs, enabling users to trace derivations incrementally. This, among other things, reveals that a constraint remains inactive either because other rules derive literals that falsify its body, or because the required literals for activation are absent.

In [6], the author introduces the notion of *minimal diagnoses* for inconsistent ELPs by identifying *conflict sets*, that is, collections of mutually incompatible constraints, such that restoring consistency amounts to disabling one constraint from each set, thereby treating constraints as candidates for removal.

The authors in [7] introduce *DWASP*, an interactive debugger for non-ground ELPs that inserts marker atoms into both the encoding and its instance and poses user queries over the fully grounded program. Constraints are tested dynamically to reveal which ones are violated, and user interaction helps isolate the rules responsible for these violations.

The authors in [8] present *SPOCK*, a declarative meta-programming technique that rewrites the original rules and constraints into a single meta-program. The answer sets of this meta-program explain why certain constraints are always or never violated. This approach relies on syntactic rewriting of the complete, ground program.

On an implementation level, constraints are transformed in such a way that their constraint violation results in an incoherent program, that is, a program without any answer sets due to cyclic dependencies. In [9], the authors propose *semi-equilibrium models*, which allow the examination of programs even in the presence of incoherence.

In contrast to these methods, which rely on constraint deletion, instance-specific grounding, or syntactic meta-rewriting, our framework operates directly on the problem encoding. It treats constraints as semantic sources of inconsistencies with derivable literals and resolves these inconsistencies via constraint-guarding λ-extensions, rather than through removal or instance-dependent diagnosis.

In [10], the authors introduce work that is most closely aligned with the topic of this paper. Their approach addresses inconsistency arising from the derivation of strongly complementary literals. To avoid confusion with other forms of inconsistency, we introduce the term *SC-inconsistency* to refer specifically to this type of inconsistency. In the cited works, such inconsistency is simply referred to as "inconsistency".

Definition 1 (SC-Consistency and SC-Inconsistency)**.** *Let $\mathcal{P}$ be an ELP. The program $\mathcal{P}$ is said to be* SC-inconsistent (strong complementarity inconsistent) *if every set $S \subseteq \mathcal{L}_{\mathcal{A}}$ satisfying (2) is inconsistent, i.e., contains a pair of strongly complementary literals l and $\bar{l}$. Otherwise, $\mathcal{P}$ is said to be SC-consistent.*

Example 3. *Suppose the following program $\mathcal{P}_3$:*

$$r_1\colon a \leftarrow c. \qquad r_2\colon \overline{a} \leftarrow d. \qquad r_3\colon \overline{a} \leftarrow {\sim}e.$$
$$r_4\colon \overline{a} \leftarrow {\sim}f. \qquad r_5\colon c. \qquad r_6\colon d.$$

Only $S = \{a, \overline{a}, c, d, \}$ satisfies condition (2). Hence, $\mathcal{P}_3$ is SC-inconsistent.

Intuitively, a program is SC-inconsistent if strongly complementary literals can be derived from $\mathcal{P}$; otherwise, it is SC-consistent.

Note that the term SC-inconsistency is not used in the original papers by Thevapalan and Kern-Isberner; we introduce it here to clearly distinguish this particular form of inconsistency from others.

A problem encoding Π is said to be *robustly SC-consistent* if it remains SC-consistent when combined with any admissible problem instance Φ, that is, any instance constructed using atoms in $\mathcal{E}(\Pi)$. In [1], the authors describe how problem encodings Π can be modified in a minimally invasive manner to achieve robust SC-consistency. To this end, they introduce the notion of *conflicts*:

Definition 2 (SC-Conflict[1]; cf. [2]). *Let $\mathcal{P}$ be an ELP containing rules r,r'. The rule set $\gamma = \{r, r'\}$ constitutes a* conflict *if:*

1. *there exists a consistent set of literals that satisfies the bodies of both r and r', and*
2. *the head literals of r and r' are strongly complementary.*

The rules in γ are referred to as conflicting *and $\Gamma_{\mathcal{P}}(r)$ denotes the set of all conflicts in $\mathcal{P}$ involving r.*

We illustrate the notion of conflicts with the following example.

Example 4. *In Example 3, $\mathcal{P}_3$ contains the conflicts $\gamma_1 = \{r_1, r_2\}$, $\gamma_2 = \{r_1, r_3\}$, and $\gamma_3 = \{r_1, r_4\}$.*

We now briefly recall the fundamental *conflict-resolution* techniques introduced in [1, 10]. Based on the notion of conflicts, a problem encoding Π is systematically modified to ensure consistency across all admissible problem instances. To this end, *conflict-resolving λ-extensions* are introduced.

We begin by recalling the characterisation of conflicts as provided in [1]:

Theorem 1 ([10]). *Let $\mathcal{P}$ be an ELP. Two rules $r, r' \in \mathcal{P}$ are conflicting if and only if the following conditions hold:*

(CP1) $H(r)$, $H(r')$ are strongly complementary;

(CP2) $B^+(r_1) \cap B^-(r_2) = \emptyset$ for $r_1, r_2 \in \{r, r'\}, r_1 \neq r_2$;

(CP3) $B^+(r) \cup B^+(r')$ is consistent.

Conditions (CP2) and (CP3) imply that such conflicts can be resolved by modifying the rule bodies to prevent their simultaneous applicability under any consistent set of literals. This motivates the notion of *conflict-preventing literals*:

Definition 3 (Conflict-Preventing Literals; cf. [1]). *Let $\mathcal{P}$ be an ELP, and let $r, r' \in \mathcal{P}$ be rules with strongly complementary head literals. Two extended literals $l^\star \in B(r), k^\star \in B(r')$ are said to be* conflict-preventing *if $atom(l^\star) = atom(k^\star)$ and $l^\star$,$k^\star$ are mutually exclusive.*

Conflict-preventing literals make it possible to resolve conflicts, as formalised in the following proposition:

Proposition 1 ([1]). *Two rules r, r' with strongly complementary head literals are non-conflicting if and only if there exist extended literals $l^\star \in B(r), k^\star \in B(r')$ such that $l^\star, k^\star$ are conflict-preventing.*

Hence, the addition of conflict-preventing literals yields rules that are no longer simultaneously applicable. This leads to the notion of *conflict-resolving λ-extensions*.

Definition 4 (λ-Extension; cf. [1]). *Let $\mathcal{P}$ be an ELP over $\mathcal{A}$, and let $r \in \mathcal{P}$. A* λ-extension *$\lambda(r)$ for r is a set of extended literals over $\mathcal{A}$ such that:*

(1) $Atom(\lambda(r)) \cap Atom(B(r)) = \emptyset$, and

(2) $Atom(\lambda(r) \cap Atom(H(r))) = \emptyset$[2].

The corresponding λ-extended rule *w.r.t. $\lambda(r)$ is:*

$$r^\star\colon H(r) \leftarrow B(r), \lambda(r). \tag{3}$$

[1] In the original literature, the term *conflict* is used. We adopt the prefix *SC-* to distinguish this notion from conflicts that are induced by constraints.

[2] Note that this condition is an additional requirement not present in the original definition of λ-extensions [1]. The inclusion of this condition ensures that the resulting λ-extensions are more informative and useful which is a feature that will prove crucial in subsequent stages of this work.

To resolve all conflicts involving r, suitable λ-extensions can be applied.

Definition 5 (Conflict-Resolving λ-Extension; cf. [1]). *Let $\mathcal{P}$ be an ELP and $r \in \mathcal{P}$ such that $\Gamma_{\mathcal{P}}(r) \neq \emptyset$. A λ-extension $\lambda(r)$ is* conflict-resolving *if for every conflict $\{r, r'\} \in \Gamma_{\mathcal{P}}(r)$, there exist extended literals $l^\star \in \lambda(r)$ and $k^\star \in B(r')$ such that $l^\star$ and $k^\star$ are conflict-preventing.*

In [1], a procedure is proposed for systematically constructing conflict-resolving λ-extensions. We continue Example 3 to illustrate the concept.

Example 5 (Example 3 contd.). *For the rule r_1 in Π_3, two conflict-resolving λ-extensions exist:*

$$\lambda_1(r_1) = \{\sim d, e, f\}, \qquad \lambda_2(r_1) = \{\overline{d}, e, f\}.$$

λ-extensions are informative, as they aim to preserve the original knowledge as far as possible, rather than simply disabling conflicting rules entirely or enforcing rigid prioritisation schemes. Instead, λ-extensions extend existing rule bodies in a minimal way, using only atoms directly related to the specific conflict. As argued in [1], traditional approaches such as semi-completion [11] can be understood as workaround strategies that syntactically suppress conflicts but fail to reflect the underlying knowledge structure. This may lead to unintended or counterintuitive answer sets, highlighting the need for more principled methods, such as λ-extensions, that address inconsistency in a constructive and transparent manner.

Example 6 ([1]). *Consider the instance $\Phi = \{pos\_test.\}$ and the encoding Π_6:*

$$\overline{p\_algy} \leftarrow \sim algy\_react. \qquad p\_algy \leftarrow pos\_test. \qquad eat\_p \leftarrow \overline{p\_algy}.$$

This program treats an individual as non-allergic if they did not experience an allergic reaction, and explicitly allergic if they tested positive. Those not allergic can eat peanuts.

According to [11], the semi-completed program Π_6' is:

$$\overline{p\_algy} \leftarrow \sim algy\_react, \sim p\_algy. \quad p\_algy \leftarrow pos\_test, \sim\overline{p\_algy}. \quad eat\_p \leftarrow \overline{p\_algy}, \sim\overline{eat\_p}.$$

One possible answer set of $\Pi_6' \cup \Phi$ is $\{pos\_test, \overline{p\_algy}, eat\_p\}$, i.e., the program predicts that an individual with a confirmed peanut allergy can eat peanuts. This counterintuitive result is clearly not intended.

Following the approach in [1], one possible outcome is $\Phi \cup \Pi_6''$ with Π_6'' given by:

$$\overline{p\_algy} \leftarrow \sim algy\_react, \sim pos\_test. \qquad p\_algy \leftarrow pos\_test. \qquad eat\_p \leftarrow \overline{p\_algy}.$$

$\Phi \cup \Pi_6''$ yields the intended answer set $\{pos\_test, p\_algy\}$.

The notion of conflicts is closely linked to SC-inconsistency. Specifically, whenever a program $\mathcal{P}$ is SC-inconsistent, then for any set S of literals satisfying (2), there exists a conflict γ in $\mathcal{P}$ such that both bodies of γ are simultaneously satisfied by S. Consequently, the absence of such conflicts guarantees that strongly complememtary literals cannot be derived.

Proposition 2 (cf. [1]). *Let Π be a problem encoding without conflicts. Then, Π is robustly SC-consistent.*

In summary, the framework proposed in [1] provides a systematic method for resolving SC-inconsistency in ELPs by identifying conflicts between rules with strongly complementary heads and applying conflict-resolving λ-extensions to prevent their simultaneous applicability. While that work focuses on inconsistencies caused by strong negation, the present paper addresses a complementary form of inconsistency–namely, *constraint-induced inconsistency*. Our goal is to establish robust consistency with respect to constraints, independently of the problem instance, by adapting the mechanics of informative λ-extensions to systematically resolve constraint-related issues at the encoding level.

4. Constraint-Induced Inconsistencies (CI-inconsistencies)

To generalise the λ-extension approach to logic programs that include constraints, we propose an approach designed to ensure CI-consistency, i.e., a notion of consistency that explicitly accounts for inconsistencies induced by constraints. The method supports practical modelling in ASP by enabling a development workflow that begins with coarse-grained rule encodings and refines them iteratively, taking into account conflicts that may arise from the addition of facts representing (future) problem instances.

In the following, we make some assumptions on the programs to be considered that help us focus on repairing CI-inconsistencies. First, we assume that programs are not SC-inconsistent (Definition 1), as such inconsistencies have already been addressed in [1] and can be resolved accordingly beforehand. Second, each rule must make sense in its own, i.e., each rule must be not contradictory and applicable to some consistent set of literals. That is, we exclude rules like $a \leftarrow \sim a.$ and $b \leftarrow a, \sim a.$ because such rules can be easily found in a preprocessing step, and fixed or removed. With regard to constraints, in this paper, we allow only positive ones, that is, constraints without default negation, consisting of at least two literals in the body. In the context of ELPs, this will allow us to deal with CI-inconsistencies in a homogeneous way, generalizing the approach presented in [1]. The treatment of CI-inconsistencies caused by general constraints also involving default negation will be left for future work.

While syntactically simple, positive constraints are semantically strict: they are violated as soon as their body is satisfied, and, due to their monotonic nature, such violations persist. This makes them particularly brittle in dynamic, instance-driven settings, where they often constitute a primary source of inconsistency.

At the same time, positive constraints are particularly well-suited for practical ASP development. They offer a declarative and intuitively interpretable way to specify global properties of desired answer sets, and we demonstrate that they align naturally with established methods such as conflict resolution via λ-extensions.

By providing a mechanism to assess whether a constraint is violated under certain instances, our approach enables domain experts to fine-tune the conditions under which specific rules apply. In this way, rules can initially be specified in coarse form, without encoding every possible exception, and refined later as needed by considering concrete possible cases to ensure consistency w.r.t. constraints.

This promotes a modular and iterative modelling paradigm: positive constraints serve not only to enforce properties of answer sets, but also as diagnostic tools for identifying and resolving problematic interactions between the encoding and instance data.

We now adapt the notion of conflicts to the setting of positive constraints. The answer sets of an ELP $\mathcal{P}$ are determined by constructing those of $\Pi_\mathcal{P} \cup \Phi_\mathcal{P}$ and verifying that they satisfy all constraints in $\Psi_\mathcal{P}$. While $\Phi_\mathcal{P} \cup \Pi_\mathcal{P}$ generates candidate answer sets, the constraints in $\Psi_\mathcal{P}$ act as filters that eliminate those violating specific conditions. Notably, constraint satisfaction can be evaluated after the solving of answer sets for the unconstrained part of the program.

Proposition 3 (cf. [12]). *Let $\mathcal{P}$ be an ELP. A set S of literals is an answer set of $\mathcal{P}$ if and only if S is an answer set of $\Pi_\mathcal{P} \cup \Phi_\mathcal{P}$ that satisfies all constraints from $\Psi_\mathcal{P}$.*

Corollary 1. *Let $\mathcal{P}$ be a program where all constraints are positive. Then, for each answer set $S \in AS(\Pi_\mathcal{P} \cup \Phi_\mathcal{P})$, we have: $S \in AS(\mathcal{P})$ if and only if there is no constraint $r_c \in \Psi_\mathcal{P}$ such that $B(r_c) \subseteq S$.*

Using these results, we can now define the notion of CI-(in)consistency:

Definition 6 (CI-Consistency). *Let $\mathcal{P}$ be a logic program comprising the problem instance $\Phi_\mathcal{P}$, the problem encoding $\Pi_\mathcal{P}$ and the set of constraints $\Psi_\mathcal{P}$. $\mathcal{P}$ is* CI-inconsistent (integrity constraint inconsistent) *if there exists at least one answer set $S = Cl((\Phi_\mathcal{P} \cup \Pi_\mathcal{P})^S)$, but each such set violates at least one constraint in $\Psi_\mathcal{P}$. Conversely, $\mathcal{P}$ is called* CI-consistent *if there exists an answer set $S = Cl((\Pi_\mathcal{P} \cup \Phi_\mathcal{P})^S)$ that satisfies all constraints from $\Psi_\mathcal{P}$.*

CI-(in)consistency is determined based on Corollary 1: fixpoints of the program without constraints are computed first, and each resulting answer set is then checked for constraint violation.

Thus, each constraint $r_c \in \Psi_\mathcal{P}$ acts as a safeguard that eliminates undesired answer sets from $\Phi \cup \Pi$. This mechanism allows the knowledge engineer to formulate general rules in $\Pi_\mathcal{P}$, while relying on constraints to rule out unintended outcomes.

Example 7 (Example 1 contd.). *Let Π_1 be the problem encoding and Ψ_1 the set of constraints as described in Example 1. Consider the problem instance $\Phi_7 = \{anem., lesn.\}$ The program $\mathcal{P}_7 = \Pi_1 \cup \Phi_7$ has the unique answer $S_7 = \{anem, lesn, leuk, mela, chem, imtx\}$. However, when the constraint in Ψ_1 is included, the resulting program $\mathcal{P}'_7 = \Pi_1 \cup \Phi_7 \cup \Psi_1$ has no answer sets because of constraint r_5, which eliminates S_7 as an answer set.*

The constraint $r_5 \in \Psi_1$ prohibits the simultaneous recommendation of chemotherapy and immunotherapy. While each treatment rule is valid in isolation, certain combinations of external atoms (e.g., facts asserting $leuk$ and $mela$) may lead to both $chem$ and $imtx$ being derived, thereby activating the constraint and eliminating the answer set. From a clinical view, such constraints are critical safeguards against harmful treatment recommendations.

To ensure that a problem encoding remains consistent across different input data, we define *admissible problem instances* as those that do not directly violate any constraint in the encoding. For example, for an encoding with the constraint '$\leftarrow a, b$', any problem instances containing both a and b are inadmissible. Specifically, while instance literals may appear in constraint bodies, each constraint must retain a sufficient dependency on the encoding. That is, constraints should not be violated merely due to the presence of certain input facts, but should require the derivation of literals through the program's rules.

Definition 7 (Admissible Problem Instance). *Given a program $\mathcal{P}$ with a set $\Psi_\mathcal{P}$ of (positive) constraints, a problem instance $\Phi_\mathcal{P}$ is said to be* admissible w.r.t. *$\Psi_\mathcal{P}$ if for each constraint $r_c \in \Psi_\mathcal{P}$ the following holds: $B(r_c) \cap \bigcup_{r \in \Phi_\mathcal{P}} H(r) = \emptyset$.*

This ensures that each constraint requires all body literals must be derived through the encoding, thereby linking constraint violations to the behaviour of the program rather than the input.

Using the notion of admissible problem instances, we define *robustly CI-consistent problem encodings*:

Definition 8 (Robust CI-Consistency). *Let Π be a problem encoding and Ψ a set of (positive) constraints. The encoding Π is referred to as* robustly CI-consistent *if, for every admissible problem instance Φ, the combined program $\Pi \cup \Phi \cup \Psi$ is CI-consistent.*

Remark: The notion of robust consistency is related to the concept of *super-coherence* [13], which guarantees that a program without strong negation admits an answer set for any extension with facts by preventing inconsistencies due to negative dependency cycles. Our approach pursues a different but complementary goal: we address inconsistencies caused by constraints and ensure CI-consistency across all admissible problem instances.

In the remainder of this paper, we demonstrate how a problem encoding can be systematically transformed to ensure robust CI-consistency across all admissible instances. Our approach builds on and extends existing methods by addressing inconsistencies that arise through the interaction of the encoding with integrity constraints, with a particular emphasis on constraint-induced violations. This complements the work in [1, 10] which focuses on resolving inconsistencies due to strongly complementary literals.

In the following, a *rule set* refers to a set of at least two rules, such that every rule in the set has a non-empty head[3]. To support several definitions introduced later in this paper, we *additionally* introduce the notion of *body-compatible* and *compatible rule sets*.

Definition 9 ((Body-)Compatible Rule Sets). *A rule set $\mathcal{R}$ is called* body-compatible *(resp.,* compatible*) if there exists a set S of literals such that*

[3]Singleton rule sets are excluded, as our approach explicitly examines the interactions between multiple rules.

(1) *S is consistent, and*

(2) *for every rule $r \in \mathcal{R}$, S satisfies $B(r)$ (resp., S satisfies $B(r) \cup H(r)$).*

Otherwise, we call $\mathcal{R}$ body-incompatible *(resp.,* incompatible*).*

Note that any compatible rule set is, in particular, body-compatible. Furthermore, any (body-)compatible rule set $\mathcal{R}$ is *downward closed*, i.e., every subset $\mathcal{R}' \subseteq \mathcal{R}$ with $|\mathcal{R}'| \geq 2$ is also (body-)compatible.

Furthermore, we obtain the following characterisation:

Theorem 2. *Let $\mathcal{R}$ be a rule set. Then $\mathcal{R}$ is compatible if and only if the following conditions hold:*

(BC1) For all rules $r, r' \in \mathcal{R}$ the sets $B^+(r)$ and $B^-(r')$ are disjoint, i.e., $B^+(r) \cap B^-(r') = \emptyset$.

(BC2) The union of the positive bodies of the rules in $\mathcal{R}$ is consistent, i.e., $\bigcup_{r \in \mathcal{R}} B^+(r)$ is consistent.

(BC3) For all rules $r, r' \in \mathcal{R}$, no head literal appears as a negative body literal in $\mathcal{R}$, i.e., $H(r) \cap B^-(r') = \emptyset$

Proof. Let $\mathcal{R} \subseteq \mathcal{P}$.
($\Rightarrow$) Assume that $\mathcal{R}$ is compatible. Then, by Definition 9, there exists a consistent set S such that S supports $r \in \mathcal{R}$. We now show that (BC1), (BC2), and (BC3) hold.

- (BC1): Assume, for contradiction that there exist $r, r' \in \mathcal{R}$ such that there exists a literal $l \in B^+(r) \cap B^-(r')$. Since S satisfies $B(r)$, we have $l \in S$. However, for S to satisfy $B(r')$, it must hold that $l \notin S$, leading to a contradiction. Hence, (BC1) holds.
- (BC2): Let $T := \bigcup_{r \in \mathcal{R}} B^+(r)$. Suppose, for contradiction, that T is inconsistent. Then there exists an atom a such that both a and $\neg a$ occur in T, implying that no consistent set S can satisfy all $B^+(r)$, contradicting the assumption that $\mathcal{R}$ is compatible. Hence, T is consistent and (BC2) holds.
- (BC3): Suppose, for contradiction, that there exist rules $r, r' \in \mathcal{R}$ such that there exists a literal $l \in H(r) \cap B^-(r')$. Since S satisfies $B(r)$, the head literal l must be in S. But since $l \in B^-(r')$, satisfying $B(r')$ requires $l \notin S$, again yielding a contradiction. Therefore (BC3) holds.

($\Leftarrow$) Conversely, suppose a rule set $\mathcal{R} \subseteq \mathcal{P}$ that satisfies (BC1), (BC2) , and (BC3). To show that $\mathcal{R}$ is compatible, define the set $S := \bigcup_{r \in \mathcal{R}} B^+(r) \cup \bigcup_{r \in \mathcal{R}} H(r)$.

Consistency of S: This follows directly from (BC2), which ensures that the union of positive bodies is consistent, and from the assumption that $\mathcal{P}$ is robustly SC-consistent.

Support of rules in $\mathcal{R}$: To show that S satisfies all rules in $\mathcal{R}$, we must verify for each $r \in \mathcal{R}$ that (S1) $B^+(r) \subseteq S$ and (S2) $B^-(r) \cap S = \emptyset$. Condition (S1) holds by construction of S, since S includes all positive body literals. For condition (S2), fix $r \in \mathcal{R}$. For any $r' \in \mathcal{R}$, we know from (BC1) that $B^+(r') \cap B^-(r) = \emptyset$, and (BC3) that $H(r') \cap B^-(r) = \emptyset$. Hence, since $S \subseteq \bigcup_{r' \in \mathcal{R}} B^+(r') \cup \bigcup_{r' \in \mathcal{R}} H(r')$, it follows that $B^-(r) \cap S = \emptyset$. Therefore, all rules in $\mathcal{R}$ are supported under the consistent set S, and thus $\mathcal{R}$ is compatible. □

The notion of (body-)compatibility is introduced to identify rule sets that may jointly violate a constraint, thereby serving as a basis for detecting potential constraint violations. This enables the application of *constraint-guarding* techniques–akin to the use of conflict-preventing literals–to strategically mitigate such interactions.

Example 8. *Suppose the following problem encoding Π_8:*

$$r_1\colon a \leftarrow d, {\sim}g. \qquad r_2\colon b \leftarrow e, {\sim}h. \qquad r_3\colon c \leftarrow f, {\sim}i.$$

All sets $\mathcal{R} \subseteq \Pi_8$ with $|\mathcal{R}| \geq 2$ are compatible, as there exists a consistent set of literals, e.g., $\{d, e, f\}$, under which all rules in $\mathcal{R}$ are supported simultaneously.

Rather than removing or modifying constraints, we examine the rules whose interactions lead to their violation. To this end, we introduce the notion of *CI-conflicts*:

Definition 10 (Constraint-Induced Conflicts (CI-Conflicts))**.** *Let Π be a problem encoding and r_c a positive constraint over $\mathcal{A}$ with at least two literals. A rule set $\mathcal{R} \subseteq \Pi$ is called a* constraint-induced conflict (CI-conflict) w.r.t. r_c *if (1) each literal in $B(r_c)$ appears as the head literal of exactly one rule in $\mathcal{R}$, and (2) $\mathcal{R}$ is compatible. The rules in an CI-conflict w.r.t. r_c are said to* violate the positive constraint r_c.

Example 9 (Example 1 contd.)**.** *Let Π_1 and Ψ_1 be defined as above. Then, the set $\{r_3, r_4\}$ is a CI-conflict w.r.t. r_5.*

CI-conflicts explain how the body of a constraint might be satisfied. Their absence guarantees that no positive constraint is violated.

Theorem 3. *Let Π be a problem encoding, and Ψ a set of positive constraints. If Π contains no CI-conflicts w.r.t. any $r_c \in \Psi$, then for any admissible problem instance Φ w.r.t. Π, we have $AS(\Pi \cup \Phi) = AS(\Pi \cup \Phi \cup \Psi)$.*

Proof. Let $\mathcal{P} := \Pi \cup \Phi$, and let $\mathcal{P}' := \Pi \cup \Phi \cup \Psi$. By Proposition 3, $AS(\mathcal{P}') = \{S \in AS(\mathcal{P}) \mid S \text{ satisfies all constraints in } \Psi\}$. Assume that Π contains no CI-conflict w.r.t. any $r_c \in \Psi$. We aim to show that under this assumption, every answer set $S \in AS(\mathcal{P})$ satisfies all constraints in Ψ.

Suppose, for contradiction, that there exists $S \in AS(\mathcal{P})$ violating some $r_c \in \Psi$. Since r_c is positive, $B(r_c) \subseteq S$. Hence, for each literal $l \in B(r_c)$, there exists a rule r with $H(r) = \{l\}$ such that r is applicable under S, i.e., r was used to derive l.

Moreover, since Φ is an admissible problem instance and contains no facts for deriving literals in the bodies of constraints, the derivation of literals in $B(r_c)$ relies solely on rules from Π and Φ. Consequently, there exists a set $\mathcal{R} \subseteq \Pi$ such that for every $l \in B(r_c)$, some $r \in \mathcal{R}$ has $H(r) = \{l\}$ and r is supported under S. The rules in $\mathcal{R}$ are compatible, implying that $\mathcal{R}$ constitutes a CI-conflict w.r.t. r_c, contradicting our assumption.

Hence, every $S \in AS(\mathcal{P})$ satisfies all constraints $r_c \in \Psi$, and thus $AS(\mathcal{P}) = AS(\mathcal{P}')$. □

As a direct consequence of Theorem 3 and Definition 10, we obtain the following characterisation:

Corollary 2. *Let Π be a problem encoding and Ψ a set of positive constraints such that Π contains no CI-conflicts w.r.t. any positive constraint $r_c \in \Psi$. Then, Π is robustly CI-consistent, that is, for any admissible problem instance Φ, the combined program $\Pi \cup \Phi \cup \Psi$ is CI-consistent.*

By modifying the rules in a CI-conflict $\mathcal{R}$ such that their bodies cannot become satisfied, unintended constraint violations can be prevented.

To introduce the notion of *constraint-guarding λ-extensions (CG λ-extensions)*, we now return to the running example:

Example 10 (Example 1 contd.)**.** *Let Π_{10} be derived from Π_1 by replacing rule r_3 with r_3': $chem \leftarrow leuk, \overline{imsp}, {\sim}mela$. Then, the program $\mathcal{P}_{10} = \Pi_{10} \cup \{\overline{imsp}, anem, lesn\} \cup \{r_5\}$ has the unique answer set $\{\overline{imsp}, anem, lesn, leuk, mela, imtx\}$.*

The modification of r_3 ensures that chemotherapy is only recommended if melanoma is not suspected. From a medical perspective, this change reflects a prioritisation of melanoma treatment. When both cancers are present–a rare but critical scenario–immunotherapy takes precedence, as it is more sensitive to treatment timing and potential interaction. By adding ${\sim}mela$, the system avoids recommending chemotherapy in such cases, preserving clinical safety.

This anticipatory adjustment disables the application of r_3 when conflicting statements arise due to (often rare, but) admissible problem instances, without compromising the intended logic of individual rules. This fine-tuning of the program helps the expert to avoid overlooking critical cases while still enabling the formalisation of their knowledge in generic (default) rules.

As a result, the program remains consistent across all admissible problem instances.

These observations lead us to the following definition.

Definition 11 (Constraint-guarding λ-Extension (CG λ-Extension))**.** *Let Π be a problem encoding, Ψ a set of constraints, and $r_c \in \Psi$ a positive constraint. Let $\mathcal{R} \subseteq \Pi$ be a CI-conflict w.r.t. r_c. A λ-extension $\lambda(r) \in \mathcal{R}$ is referred to as a* constraint-guarding λ-extension (CG λ-extension) w.r.t. r_c *if there exist two extended literals $l^\star$ in $\lambda(r)$ and $k^\star$ in some rule $r' \in \mathcal{R}$ such that $l^\star, k^\star$ are mutually exclusive. The extended literal $l^\star \in \lambda(r)$ is then referred to as a* guard literal*.*

Lemma 1 (Condition for Existence of Constraint-Guarding λ-Extensions)**.** *Let r be a rule in an ELP $\mathcal{P}$, and suppose there exists a CI-conflict $\mathcal{R}$ w.r.t. some positive constraint such that $r \in \mathcal{R}$. The exists a CG λ-extension $\lambda(r)$ if there is a rule $r' \in \mathcal{R}$ and an extended literal $l^\star \in B(r') \setminus B(r)$ such that $atom(l^\star) \notin Atom(B(r))$, and $atom(l^\star) \notin Atom(H(r))$.*

Proof. We prove the contrapositive: Suppose that for all $r' \in \mathcal{R}$ and all $l^\star \in B(r') \setminus B(r)$, either $atom(l^\star) \notin Atom(B(r))$, or $atom(l^\star) \notin Atom(H(r))$. Then no CG λ-extension $\lambda(r)$ exists.

Assume that for every $r' \in \mathcal{R}$, every extended literal $l^\star \in B(r') \setminus B(r)$ satisfies $atom(l^\star) \in Atom(B(r)) \cup Atom(H(r))$. This implies there exists no literal in $B(r') \setminus B(r)$ whose atom is disjoint from both the body and the head of r.

Suppose we attempt to construct a CG λ-extension $\lambda(r)$. To do so, we must add a guard literal $\sim l^\star$ (alternatively $\overline{l^\star}$) to the body of r, aiming to make r *body-incompatible* with some $r' \in \mathcal{R}$ while leaving r' unaffected. This requires that $l^\star$ occurs in $B(r')$ but not in $B(r)$, and $atom(l^\star)$ does not appear in $B(r) \cup H(r)$, to ensure $\sim l^\star$ is a valid guard and does not interfere with the existing semantics of r'. However, by our assumption, every candidate literal $l^\star \in B(r') \setminus B(r)$ fails this requirement as its atom is already present in r. Hence, any attempt to use $\sim l^\star$ as a guard literal either introduces redundancy, or creates self-blocking behaviour (if the atom occurs in $H(r)$), or results in no incompatibility with r' at all. Consequently, no valid guard literal exists to distinguish r from r', and thus no constraint-guarding λ-extension $\lambda(r)$ exists. This completes the contraposition and proves the lemma. □

Given an encoding Π with a CI-conflict $\mathcal{R}$ w.r.t. a positive constraint, we say $\mathcal{R}$ is *guardable* if there exists a CG λ-extension for $\mathcal{R}$.

Constraint-guarding λ-extensions allow for the construction of problem encodings that remain consistent across all admissible instances.

Proposition 4. *Let Π be a problem encoding, Ψ a set of constraints. Let $r_c \in \Psi$ be a positive constraint, and let $\mathcal{R} \subseteq \Pi$ be a CI-conflict w.r.t. r_c. Suppose $r \in \mathcal{R}$, and let $\lambda(r)$ be a CG λ-extension of r w.r.t. r_c. Let r'' be the resulting λ-extended rule w.r.t. $\lambda(r)$. Then, the set $\mathcal{R} \setminus \{r\} \cup \{r''\}$ is no longer a CI-conflict w.r.t. r_c.*

Proof. Assume that $\mathcal{R} \in \Pi$ is a CI-conflict w.r.t. a positive constraint r_c. By Definition 10, this implies that $\mathcal{R}$ is compatible. Let $r \in \mathcal{R}$ and let $\lambda(r)$ be a CG λ-extension for r w.r.t. r_c. Denote by r'' the λ-extended rule w.r.t. $\lambda(r)$, and define $\mathcal{R}' := \mathcal{R} \setminus \{r\} \cup \{r''\}$. By Definition 11, $\lambda(r)$ is constraint-guarding in $\mathcal{R}'$, implying that $\mathcal{R}'$ is not body-compatible Consequently, $\mathcal{R}'$ is not compatible and therefore not a CI-conflict w.r.t. r_c. □

Hence, robust CI-consistency can be achieved in a problem encoding by iteratively resolving all CI-conflicts through appropriate constraint-guarding λ-extensions.

Proposition 5. *Let Π_0 be a (potentially CI-inconsistent) problem encoding and and let Ψ be a set of positive constraints. Furthermore, let all CI-conflicts in Π_0 be guardable. Then there exists a finite sequence of encodings $\Pi_0, \Pi_1, \ldots, \Pi_n$ such that for each $i < n$:*

- *a CI-conflict $\mathcal{R} \subseteq \Pi_i$ w.r.t. some positive constraint $r_c \in \Psi$ is identified, and*
- *Π_{i+1} is obtained from Π_i by applying a CG λ-extension for a rule $r_i \in \mathcal{R}$ w.r.t. r_c to r_i, yielding a modified rule r_i' that weakens r_i w.r.t. the constraint-induced inconsistencies.*

This process terminates after finitely many steps. The final encoding Π_n is then robustly CI-consistent, meaning that no constraint in Ψ can be violated under any admissible instance, and that all such constraint-induced inconsistencies have been resolved without the removal of any rule from the original encoding.

Proof. We proceed by induction over the CI-conflicts in Π w.r.t. some positive constraint in Ψ.

Let $\Pi_0 = \Pi$. If Π_0 is already robustly CI-consistent, we are done.

Otherwise, there exists a constraint $r_c \in \Psi$ and a CI-conflict $\mathcal{R} \subseteq \Pi_0$ such that the rules in $\mathcal{R}$ collectively allow a derivation of a set of literals that violate r_c under some admissible instance. By assumption, not all rules in $\mathcal{R}$ share the same body, so at least one rule $r_i \in \mathcal{R}$ admits a CG λ-extension $\lambda(r_i)$ for r_i w.r.t. r_c.

We apply $\lambda(r_i)$ to weaken r_i w.r.t. the derivation that violates r_c, obtaining r_i'. The next encoding is defined as $\Pi_1 = \Pi_0 \setminus \{r_i\} \cup \{r_i'\}$.

This modification eliminates the particular potential constraint violation associated with the derivation of r_i. Moreover, because the λ-extension reduces the derivational strength of r_i, it cannot reintroduce the same conflict.

This process is repeated iteratively. At each step i, we select a remaining unresolved CI-conflict $\mathcal{R} \subseteq \Pi_i$ and choose a rule $r_i \in \mathcal{R}$ that can be individually weakened due to the assumption that $\mathcal{R}$ is guardable. This guarantees progress by resolving at least one distinct conflict per step.

Since the total number of conflicts is finite (due to the finiteness of Π, Ψ, and possible derivations), the process must terminate after finitely many steps.

Let Π_n be the final encoding. By construction, it is robustly CI-consistent w.r.t. all constraints in Ψ (cf. Corollary 2), and every rule from the original encoding is either unchanged or replaced by a weaker version (i.e., extended rule body), without any deletions. □

Algorithm 1 Constructing λ-Extensions for Guardable CI-conflicts

Require: Encoding $\Pi_\mathcal{P}$, constraint set $\Phi_\mathcal{P}$, guardable CI-conflict $\mathcal{R} \subseteq \Pi_\mathcal{P}$ w.r.t. $r_c \in \Psi_\mathcal{P}$
Ensure: λ-extension $\lambda(r)$ for some $r \in \mathcal{R}$
1: Select two distinct rules $r, r' \in \mathcal{R}$
2: $X \leftarrow B(r') \setminus B(r)$
3: $Y \leftarrow \{l^\star \in X \mid atom(l^\star) \notin B(r), atom(l^\star) \notin H(r)\}$
4: **if** $Y = \emptyset$ **then**
5: **abort:** no valid extended literal found
6: **end if**
7: Select $l^* \in Y$
8: Define $\lambda_1(r) := \{\sim l^\star\}$ and $\lambda_2(r) := \{\overline{l^\star}\}$
9: Construct two λ-extended rules w.r.t. r:
10: $r^\sim := H(r) \leftarrow B(r), \lambda_1(r).$
11: $r^\neg := H(r) \leftarrow B(r), \lambda_2(r).$
12: Select either $r^\sim$ or $r^\neg$ to replace r in $\Pi_\mathcal{P}$

To systematically enforce incompatibility within a CI-conflict, Algorithm 1 constructs a CG λ-extension for a selected rule $r \in r'$, where $\mathcal{R} \subseteq \Pi$ is a guardable CI-conflict. The procedure begins by selecting a second rule $r' \in \mathcal{R}$ that the expert intends to render incompatible with r (line 1). The algorithm then identifies literals $l^\star \in B(r') \setminus B(r)$ that do not appear in the head or body of r (lines 2– 3). These literals form the candidate set Y, from which the expert selects a literal $l^\star$ (line 7). Based on this choice, the algorithm constructs two possible λ-extensions: $\lambda_1(r) = \{\sim l^\star\}$, and $\lambda_2(r) = \{\overline{l^\star}\}$ (line 8). These generate two λ-extended rule variants of r, denoted by $r^\sim$ and $r^\neg$, which respectively enforce incompatibility with r' via strong or default negation of $l^\star$ (lines 10–11). The expert then selects one of these λ-extended rules to replace the original rule r (line 12). This process ensures that any resulting answer set avoids satisfying the bodies of both rules simultaneously, thereby resolving the potential constraint violation through strategic weakening.

Let us illustrate Algorithm 1 using the following example:

Example 11. *Let Π_{11} be a problem encoding consisting of the following rules:*

$$r_1\colon a \leftarrow d, \sim e. \qquad r_2\colon b \leftarrow f. \qquad r_3\colon c \leftarrow g, \sim h. \qquad r_4\colon i \leftarrow k, \bar{l}. \qquad r_5\colon j \leftarrow l, \sim m.$$

Additionally, consider the following set Ψ_{11} of constraints:

$$r_6\colon a, b, c. \qquad\qquad r_7\colon i, j.$$

The encoding Π_{11} contains two CI-conflicts: $\mathcal{R}_1 = \{r_1, r_2, r_3\}$ w.r.t. r_6 and $\mathcal{R}_2 = \{r_4, r_5\}$ w.r.t. r_7.

Since the rules within each CI-conflicts do not share any atoms, both CI-conflicts are guardable.

To resolve the potential constraint violation in $\mathcal{R}_1$, the expert selects the rule pair r_1, r_2, and extends r_1. We define $X_1 = B(r_2) \setminus B(r_1) = \{f\}$, and thus obtain $Y_1 = \{f\}$. Selecting f, we construct possible λ-extensions for r_1, namely $\lambda_{1,1} = \{\sim f\}$, and $\lambda_{1,2} = \{\overline{f}\}$. The corresponding λ-extended rules for r_1 are:

$$r_1^{\sim}\colon a \leftarrow d, \sim e, \sim f. \qquad\qquad r_1^{\neg}\colon a \leftarrow d, \sim e, \overline{f}.$$

Similarly, for the CI-conflict $\mathcal{R}_2$, we choose to extend r_4. We define $X_2 = B(r_5) \setminus B(r_4) = \{l, \sim m\}$, resulting in $Y_2 = \{\sim m\}$. Selecting $\sim m$, we construct possible λ-extensions for r_4, namely, $\lambda_{2,1} = \{m\}$, and $\lambda_{2,2} = \{\sim\overline{m}\}$. The corresponding λ-extended rules for r_4 are:

$$r_4^{\sim}\colon i \leftarrow k, m. \qquad\qquad r_4^{\neg}\colon i \leftarrow k, \sim\overline{m}.$$

Suppose the expert selects $r_1^{\sim}$: and $r_4^{\neg}$: as the λ-extended rules. We then obtain the following modified problem encoding Π'_{11}:

$$r_1^{\sim}\colon a \leftarrow d, \sim e, \sim f. \qquad r_2\colon b \leftarrow f. \qquad r_3\colon c \leftarrow g, \sim h. \qquad r_4^{\neg}\colon i \leftarrow k, \overline{m}. \qquad r_5\colon j \leftarrow l, \sim m.$$

Since Π'_{11} contains no CI-conflicts, the modified program is CI-consistent w.r.t. Ψ_{11}.

In summary, our approach provides a constructive method for ensuring robust CI-consistency. By systematically detecting CI-conflicts and resolving them through suitable constraint-guarding and informative λ-extensions, we isolate the sources of potential constraint violations and weaken only the necessary rules. This targeted repair strategy preserves the original encoding structure to the greatest extent possible while guaranteeing that all constraints remain satisfied under any admissible instance. As a result, the final encoding satisfies robust CI-consistency without requiring rule deletion or instance-specific modifications.

5. Conclusion and Outlook

In this paper, we have introduced a novel framework for managing CI-conflicts in ELPs. By allowing users to specify coarse-grained conditions and constraints directly at the encoding level, our method automatically determines which constraints could be violated under some (unspecified) instance. We isolate the culprit rules whose derivations risk violating a constraint, leveraging the established concept of constraint-guarding λ-extensions, originally defined for achieving SC-consistency. Applying these λ-extensions yields robust CI-consistency, meaning that all constraints remain satisfied without deleting rules or depending on particular data instances.

Our main contributions are

(1) a formal definition of CI-conflicts as a semantic interaction between constraints and potentially derivable literals;

(2) an algorithm for detecting and characterising such conflicts at the encoding level; and

(3) a constructive repair procedure based on λ-extensions that minimally modifies only those rules necessary to restore consistency w.r.t. constraints.

For future work, we plan to generalise our inconsistency analysis framework to accommodate all forms of constraints, including those involving default negation, thereby extending λ-extensions to all CI-conflicts. Furthermore, we are currently working on relaxing the current restrictions on admissible problem instances–specifically, by allowing instances to contain literals that appear in constraints.

In parallel, we are developing a complementary theory of consistency w.r.t. odd cycles for ground programs. Once this theory is established, we will investigate whether and how robust consistency w.r.t. odd cycles can be achieved. Through these developments, we aim to provide a unified, instance-independent methodology for diagnosing and repairing all types of inconsistencies in ASP encodings.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] A. Thevapalan, G. Kern-Isberner, On establishing robust consistency in answer set programs, Theory Pract. Log. Program. 23 (2023) 1094–1127. doi:10.1017/S1471068422000357.

[2] A. Thevapalan, G. Kern-Isberner, Sorting strategies for interactive conflict resolution in ASP, in: E. Pontelli, S. Costantini, C. Dodaro, S. A. Gaggl, R. Calegari, A. S. d'Avila Garcez, F. Fabiano, A. Mileo, A. Russo, F. Toni (Eds.), Proceedings 39th International Conference on Logic Programming, ICLP 2023, Imperial College London, UK, 9th July 2023 - 15th July 2023, volume 385 of *EPTCS*, 2023, pp. 116–128. doi:10.4204/EPTCS.385.13.

[3] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (1991) 365–386.

[4] C. Sakama, K. Inoue, Paraconsistent stable semantics for extended disjunctive programs, J. Log. Comput. 5 (1995) 265–285. doi:10.1093/LOGCOM/5.3.265.

[5] J. Oetsch, J. Pührer, H. Tompits, Stepwise debugging of answer-set programs, Theory Pract. Log. Program. 18 (2018) 30–80. doi:10.1017/S1471068417000217.

[6] T. Syrjänen, Debugging inconsistent answer set programs, in: The 11th International Workshop on Nonmonotonic Reasoning, Low Wood hotel, Lake District, England, UK, 30 May - 1 June, 2006, number IfI-06-04 in Clausthal University of Technology, Department of Informatics, Technical Report, Clausthal University of Technology, Germany, 2006, pp. 77–83.

[7] C. Dodaro, P. Gasteiger, K. Reale, F. Ricca, K. Schekotihin, Debugging non-ground ASP programs: Technique and graphical tools, Theory Pract. Log. Program. 19 (2019) 290–316. doi:10.1017/S1471068418000492.

[8] M. Gebser, J. Pührer, T. Schaub, H. Tompits, A meta-programming technique for debugging answer-set programs, in: D. Fox, C. P. Gomes (Eds.), Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008, AAAI Press, 2008, pp. 448–453.

[9] T. Eiter, M. Fink, J. Moura, Paracoherent answer set programming, in: Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning, KR'10, AAAI Press, Toronto, Ontario, Canada, 2010, p. 486–496.

[10] A. Thevapalan, G. Kern-Isberner, Towards interactive conflict resolution in ASP programs, in: M. Martinez, I. Varcinczak (Eds.), Proceedings of the 18th International Workshop on Non-Monotonic Reasoning, NMR 2020, 2020, pp. 29–36.

[11] M. Caminada, Well-founded semantics for semi-normal extended logic programs, in: Proc. 11th Int'l Workshop on Nonmonotonic Reasoning, 2006, pp. 103–108.

[12] M. Gelfond, Y. Kahl, Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach, Cambridge University Press, 2014. doi:10.1017/CBO9781139342124.

[13] M. Alviano, W. Faber, S. Woltran, Complexity of super-coherence problems in ASP, Theory and Practice of Logic Programming 14 (2013) 339–361. doi:10.1017/s147106841300001x.