Inconsistency Measures for Disjunctive Logic Programs Under Answer Set Semantics

Markus Ulbricht[1], Matthias Thimm[2],
Gerhard Brewka[1]

[1]Department of Computer Science
Leipzig University, Germany
{brewka,mulbricht}@informatik.uni-leipzig.de,
[2]Institute for Web Science and Technologies
University of Koblenz-Landau, Germany
thimm@uni-koblenz.de

Abstract

We address the issue of quantitatively assessing the severity of inconsistencies in disjunctive logic programs under the answer set semantics. Taking the non-monotonicity of answer set semantics into account brings new challenges that have to be addressed by reasonable accounts of inconsistency measures. We investigate the behaviour of inconsistency in logic programs by revisiting existing rationality postulates for inconsistency measurement and developing novel ones taking non-monotonicity into account. Further, we develop new measures for this setting and investigate their properties, in particular with respect to their compliance to these rationality postulates and their computational complexity.

1 Introduction

Inconsistency is an omnipresent phenomenon in logical accounts of knowledge representation and reasoning (KR) [9, 27, 23, 16, 12]. Classical logics usually suffer from the *principle of explosion* which renders reasoning meaningless, as everything can be derived from inconsistent theories. Therefore, reasoning under inconsistency [4, 32, 34] is an important research area in KR. In general, one can distinguish two paradigms in handling inconsistent information. The first paradigm advocates living with inconsistency but providing non-classical semantics that allow the derivation of non-trivial information, such as using paraconsistent reasoning [7], reasoning with possibilistic logic [16, 17], or formal argumentation [3]. The second paradigm is about explicitly restoring con-

sistency, thus changing the theory itself, as it is done in e. g. belief revision [27] or belief merging [33]. A quantitative approach for *analyzing* inconsistencies is given by the field *inconsistency measurement* which investigates functions $\mathcal{I}$ that assign real numbers to knowledge bases, with the intuitive meaning that larger values indicate more severe inconsistency, see [46, 45, 48] for surveys and [47, 30, 26, 6, 1, 40] for some recent approaches.

Answer set programming (ASP, see [10] for an overview) is an emerging problem solving paradigm. It is based on logic programs under the answer set semantics [22, 21], a popular non-monotonic formalism for knowledge representation and reasoning which consists of rules possibly containing default-negated literals. Inconsistencies occur in ASP for two reasons, cf. [43]. First, the rules allow the derivation of two complementary literals l and $\neg l$—also called *incoherence* in e. g. [37]—thus producing inconsistencies similar to e. g. propositional logic. Second, due to the use of default negation it may happen that some literal assumed to be false is again derived (called *instability*). Hence, analyzing and handling inconsistency in ASP poses additional challenges (in comparison to monotonic logics) that need to be addressed, cf. [19, 13]. Some few works handle these challenges by adapting the classical techniques mentioned above to ASP, such as paraconsistent reasoning [8] or belief revision [15].

In this paper, we investigate the problem of measuring inconsistency in ASP. The issue of measuring inconsistency in logic programs is more challenging compared to the setting of propositional knowledge bases due to the non-monotonicity of answer set semantics. This becomes apparent when considering the *Monotonicity* postulate which is usually satisfied by inconsistency measures for propositional knowledge bases. It demands $\mathcal{I}(P') \geq \mathcal{I}(P)$ whenever $P \subseteq P'$ for any logical theories P and P', i. e., the severity of inconsistency cannot be decreased by adding new information. Consider now the two logic programs P and P' given as follows:

$$P: \begin{array}{l} b \leftarrow \text{not } a. \\ \neg b \leftarrow \text{not } a. \end{array} \qquad P': \begin{array}{l} b \leftarrow \text{not } a. \\ \neg b \leftarrow \text{not } a. \\ a. \end{array}$$

We have $P \subseteq P'$ but P is inconsistent while P' is not, so we would expect $\mathcal{I}(P') < \mathcal{I}(P')$ for any reasonable measure $\mathcal{I}$. Therefore, simply taking classical inconsistency measures and applying them to the setting of logic programs does not yield the desired behavior. Many rationality postulates such as *Monotonicity* are already disputed in the case of propositional knowledge bases, cf. [5]. Taking non-monotonicity of the knowledge representation formalism into account, a rational account of the severity of inconsistency calls for a specific investigation, which we will undertake in the remainder of this paper.

The main contributions of this paper are as follows:

1. We revisit the notion of inconsistency measures for ASP and develop

six new inconsistency measures for this setting (Section 3). Several of the measures are based on the effort it takes to repair an inconsistent program, others measure various kinds of distances between the actual and the intended outcome.

2. We critically examine existing rationality postulates, and develop novel ones taking non-monotonicity into account (Section 4). Several of our postulates aim to replace the unintended monotonicity postulate by weaker variants.

3. We analyze our new measures by checking their compliance with the rationality postulates (Section 5). In a nutshell, the main outcome of this analysis is that our new measures are well-behaved in the light of the postulates.

4. We finally perform an in-depth complexity analysis of computational problems related to our measures (Section 6).

Furthermore, we will give necessary preliminaries in Section 2 and conclude in Section 7. Readers familiar with logic programming and thus tempted to skip the preliminaries section should be aware that—for reasons explained in Section 2—we use a slightly nonstandard definition of answer sets which allows programs to have multiple inconsistent answer sets.

A brief description of a proper subset of the postulates and measures investigated in this paper was presented in the extended abstract [50]. The abstract did not cover disjunctive programs, and no complexity analysis was given.

2 Preliminaries

In this paper, we consider logic programs with disjunction in the head of rules and two kinds of negation, namely strong negation "¬" and default negation "not", under the answer set semantics [22, 21]. In [22] such programs were called extended disjunctive databases, whereas Gelfond and Leone [21] simply speak of logic programs or A-Prolog programs. We will call these programs *extended disjunctive logic programs*.

For the remainder of this paper, we assume we are (implicitly) given an infinite set $\mathcal{L}$ of literals. Now, an extended disjunctive logic program P is a finite set of rules r of the form

$$l_0 \vee \ldots \vee l_k \leftarrow l_{k+1}, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n. \tag{1}$$

where $l_0, \ldots, l_n \in \mathcal{L}$ and $0 \leq k \leq m \leq n$. In particular, no function symbols or variables occur in r.

For a rule r of the form (1) we write $head(r) = \{l_0, \ldots, l_k\}$, $body(r) = \{l_{k+1}, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n\}$, $pos(r) = \{l_{k+1}, \ldots, l_m\}$ and $neg(r) =$

$\{l_{m+1}, \ldots, l_n\}$. For a set M of literals, let $\mathcal{A}(M)$ be the set of all atoms occurring in M. We let $\mathcal{A}(r)$ and $\mathcal{L}(r)$ be the set of all atoms and literals occurring in r, respectively. Similarly, let $\mathcal{A}(P)$ and $\mathcal{L}(P)$ be the set of all atoms and literals that occur in a program P, respectively. Further, let $body(P) = \cup_{r \in P} body(r)$, and analogously for $pos(P)$ and $neg(P)$.

We write "$l_0 \vee \ldots \vee l_k$." instead of "$l_0 \vee \ldots \vee l_k \leftarrow$." for rules with a trivial body. If in addition $k = 0$ holds, i. e., the rule is of the form "l_0.", we call it a *fact*.

The set of all disjunctive logic programs is denoted by $\mathcal{P}$. Throughout the paper, we distinguish the following subclasses of $\mathcal{P}$:

- *Extended disjunctive logic programs*, i. e., programs that consist of rules of the form (1). Since we do not particularly focus on programs without the occurrence of strong negation "$\neg$", we will call such programs simply *disjunctive logic programs* in most cases.

- *Extended logic programs*, i. e., programs that consist of rules of the form (1) with $k = 0$, i. e., with no occurrence of the disjunction "$\vee$" in the head of rules. In order to emphasize the lack of disjunction, we will also call such programs *extended disjunction-free logic programs* or *disjunction-free logic programs* for short. Analogously, we call rules of the form (1) with $k = 0$ *disjunction-free rules*.

- *Extended disjunctive classical logic programs*, i. e., programs that consist of rules of the form (1) with $n = m$, i. e., with no occurrence of the default negation "not" in the body of rules. We will call such programs simply *classical logic programs*. Analogously, we call rules of the form (1) with $m = n$ *classical rules*.

- *Extended classical disjunction-free logic programs*, i. e., programs that consist of rules of the form (1) with $k = 0$ and $n = m$, i. e., neither "$\vee$" nor "not" occurs in a rule. We will call such programs simply *classical disjunction-free logic programs*. Analogously, we call rules of the form (1) with $m = n$ *classical disjunction-free rules*.

We now turn to the semantics, i. e., the definition of answer sets. There are actually variants of the definition in the literature which differ in whether inconsistent answer sets are admitted or not. The original definition in [22] allows for a single inconsistent answer set, namely $\mathcal{L}$, in cases where a subset of rules without default negation generates an inconsistency. In this paper, we are interested in more fine-grained distinctions than the single inconsistent answer set $\mathcal{L}$ would allow. For this reason answer sets in this paper can be arbitrary subsets of $\mathcal{L}$. For a set M of literals and a literal l we say M satisfies l ($M \models l$) iff $l \in M$. If L is a set of literals, then $M \models L$ iff $M \models l$ for all $l \in L$. Now consider a classical rule, i. e., a rule r of the form (1) with $m = n$. We

say M satisfies r, denoted by $M \models r$ iff $\exists i \in \{0, \ldots, k\} : M \models l_i$ whenever $M \models body(r)$.

Now we are ready to define answer sets of a given program.

Definition 1. Let P be a classical disjunctive logic program. A set M of literals is called an *answer set* of P if $M \models r$ for all rules $r \in P$ (denoted by $M \models P$) and there is no $M' \subsetneq M$ with $M' \models P$. For an arbitrary program P, M is an answer set of P iff M is an answer set of P^M, where P^M is the *reduct* of P with respect to M, i. e.,

$$P^M = \{head(r) \leftarrow pos(r) \mid r \in P,\ neg(r) \cap M = \emptyset\}.$$

Note that, so far, we defined what an *answer set* is no matter whether it is consistent or not. In principle, *any* set of literals can be an answer set of a given program. We now distinguish consistent and inconsistent answer sets.

Definition 2. A set M of literals is called *consistent* if it does not contain both a and $\neg a$ for an atom a. A program P is called *consistent* if it has at least one consistent answer set, otherwise it is called *inconsistent*. Let $\mathrm{Ans}(P)$ denote the set of all answer sets of P and $\mathrm{Ans}_{Inc}(P)$ and $\mathrm{Ans}_{Con}(P)$ the inconsistent and consistent ones, respectively. So we have $\mathrm{Ans}(P) = \mathrm{Ans}_{Inc}(P) \cup \mathrm{Ans}_{Con}(P)$.

Hence, a program P can be inconsistent, because

- it has no answer set, i. e., $\mathrm{Ans}(P) = \emptyset$ or
- it only has inconsistent answer sets, i. e., $\mathrm{Ans}(P) = \mathrm{Ans}_{Inc}(P)$.

Note that in particular, P is inconsistent iff $\mathrm{Ans}_{Con}(P) = \emptyset$.

Example 1. The program

$$P_1 : \qquad a \vee \neg a. \qquad\qquad \neg a \leftarrow a.$$

has two answer sets, $\{a, \neg a\}$ and $\{\neg a\}$. The latter is consistent and so is the program. The same program with "$a \leftarrow \neg a$." as additional rule, i. e., the program

$$P_2 : \qquad a \vee \neg a. \qquad\qquad \neg a \leftarrow a. \qquad\qquad a \leftarrow \neg a.$$

has $\{a, \neg a\}$ as unique answer set and is therefore inconsistent. Now consider the program

$$P_3 : \qquad b \leftarrow \text{not } c. \qquad\qquad c \leftarrow \text{not } d. \qquad\qquad d \leftarrow \text{not } b.$$

One can check that P_3 does not have an answer set. In fact, it is a quite simple example of an inconsistency that stems from an odd loop in the dependency graph (cf. Definition 15 below). In contrast,

$$P_3' : \qquad b \leftarrow \text{not } c. \qquad\qquad c \leftarrow \text{not } d. \qquad\qquad d \leftarrow \text{not } b.$$
$$\qquad\qquad d \leftarrow \text{not } e.$$

has $\{b, d\}$ as answer set, because d can inferred due to the added rule "$d \leftarrow$ not e.". However, the program

$$P_3'': \quad b \leftarrow \text{not } c. \qquad c \leftarrow \text{not } d. \qquad d \leftarrow \text{not } b.$$
$$d \leftarrow \text{not } e. \qquad e.$$

again has no answer set.

3 Inconsistency Measures

In the literature on inconsistency measurement—see e. g. [28, 24, 46]—inconsistency measures are functions that aim at assessing the severity of the inconsistency in knowledge bases formalized in propositional logic. Here, we are interested in measuring inconsistency for logic programs and only consider measures defined on those. Let $\mathbb{R}^{\infty}_{\geq 0}$ be the set of non-negative real values including ∞.

Definition 3. An *inconsistency measure* $\mathcal{I}$ is a function $\mathcal{I} : \mathcal{P} \to \mathbb{R}^{\infty}_{\geq 0}$.

The basic intuition behind an inconsistency measure $\mathcal{I}$ is that the larger the inconsistency in P the larger the value $\mathcal{I}(P)$. We now propose concrete inconsistency measures for logic programs. Inconsistency of logic programs can occur due to two different reasons, namely because the program has no answer set at all or because all answer sets are inconsistent, cf. [43]. Different measures should assess those reasons differently.

We start with a very simple measure which just indicates whether a given program is consistent or not [29].

Definition 4. Define $\mathcal{I}_{01} : \mathcal{P} \to \{0, 1\}$ via

$$\mathcal{I}_{01}(P) = \begin{cases} 0 & \text{if } P \text{ is consistent,} \\ 1 & \text{otherwise} \end{cases}$$

for all $P \in \mathcal{P}$.

We call $\mathcal{I}_{01}$ the *drastic inconsistency measure*. Of course, this measure fails to provide the distinction we are aiming for, namely the distinction between less and more inconsistent programs. We will therefore introduce various more fine-grained measures in the remainder of this section.

In Section 3.1, we introduce a generalization of the measure $\mathcal{I}_{\mathsf{MI}}$ [29] which, in its original definition, counts the number of minimal inconsistent subsets of a knowledge base $\mathcal{K}$ (a set of propositional formulas). For that, we utilize a notion of inconsistency for nonmonotonic logics developed in [11]. We continue with measures that are based on the distance of inconsistent answer sets to consistent ones (Section 3.2). Then, we consider syntactic approaches that are based on the effort needed to turn an inconsistent program into a consistent one (Section 3.3).

3.1 Measures based on Strong Inconsistency

One of the most basic inconsistency measures for a propositional knowledge base $\mathcal{K}$ is $\mathcal{I}_{\mathsf{MI}}$ [29] defined via $\mathcal{I}_{\mathsf{MI}}(\mathcal{K}) = |\mathsf{MI}(\mathcal{K})|$ where $\mathsf{MI}(\mathcal{K})$ is the set of minimal inconsistent subsets of $\mathcal{K}$. As consistent programs may contain inconsistent subsets—cf. e. g., P_3' from above—this measure is quite meaningless in ASP. In [11], a refined notion of inconsistency for nonmonotonic logics has been proposed which does not have this drawback. We give the definition for the special case of ASP.

Definition 5. Let P be a logic program. A subset $H \subseteq P$, is called *strongly P-inconsistent* if $H \subseteq H' \subseteq P$ implies H' is inconsistent. The set H is *minimal strongly P-inconsistent* if H is strongly P-inconsistent and $H' \subsetneq H$ implies that H' is not strongly P-inconsistent. Let $SI_{min}(P)$ be the set of all minimal strongly P-inconsistent subsets of P.

The main motivation for this notion of inconsistency is that a generalization of Reiter's hitting set duality [42] can be proved (cf. [11] for more details). Now, we can define our generalized measure as follows.

Definition 6. Define $\mathcal{I}_{\mathsf{MSI}} : \mathcal{P} \to \mathbb{R}^{\infty}_{\geq 0}$ via

$$\mathcal{I}_{\mathsf{MSI}}(P) = |SI_{min}(P)|$$

i. e., the measure outputs the number of minimal strongly P-inconsistent subsets of P.

Given a propositional knowledge base $\mathcal{K}$, the minimal inconsistent subsets $\mathsf{MI}(\mathcal{K})$ are interpreted as the "raw" conflicts within $\mathcal{K}$. Similarly, $\mathcal{I}_{\mathsf{MSI}}$ counts the number of "raw" conflicts within a program P.

Example 2. Consider

$$P_2 : \qquad a \vee \neg a. \qquad\qquad \neg a \leftarrow a. \qquad\qquad a \leftarrow \neg a.$$

again. Any proper subset H of the program is consistent. Hence, P_2 itself is the only strongly P_2-inconsistent subset. We obtain

$$\mathcal{I}_{\mathsf{MSI}}(P_2) = 1.$$

Now consider

$$\begin{array}{lllll} P_3'' : & b \leftarrow \text{not } c. & c \leftarrow \text{not } d. & & d \leftarrow \text{not } b. \\ & d \leftarrow \text{not } e. & e. & & \end{array}$$

The subset

$$H = \{b \leftarrow \text{not } c., \quad c \leftarrow \text{not } d., \quad d \leftarrow \text{not } b.\}$$

is inconsistent, but it is *not* strongly P_3''-inconsistent as it is contained in the consistent set

$$P_3': \quad b \leftarrow \text{not } c. \qquad c \leftarrow \text{not } d. \qquad d \leftarrow \text{not } b. \qquad d \leftarrow \text{not } e.$$

One can verifiy that

$$SI_{min}(P_3'') = \{\{b \leftarrow \text{not } c.,\ c \leftarrow \text{not } d.,\ d \leftarrow \text{not } b.,\ e.\}\}$$

and hence,

$$\mathcal{I}_{\mathsf{MSI}}(P_3'') = 1.$$

3.2 Distance-based Measures

We now consider measures that focus on the (possibly inconsistent) answer sets of a program instead of the program itself. To this end, we now introduce inconsistency measures that make use of distance measures to assess the inconsistency of answer sets. Note that distance-based measures have also been used in the setting of propositional logic, see for instance [26].

Observe that, while a program P might not have an answer set, for any set M of literals the reduct P^M is a classical logic program and thus has a nonempty set of answer sets. So, one could consider the distance between a set M and consistent answer sets in $\text{Ans}_{Con}(P^M)$.

Definition 7. A mapping $d : 2^{\mathcal{L}} \times 2^{\mathcal{L}} \to [0, \infty)$ is called a *distance* if it satisfies

- $d(X, Y) = 0$ if and only if $X = Y$,
- $d(X, Y) = d(Y, X)$,
- $d(X, Y) \leq d(X, Z) + d(Z, Y)$

for any $X, Y, Z \subseteq \mathcal{L}$.

In the following, we only consider the number of literals in the symmetric difference of two sets as an example of a distance measure between sets. Investigating other distances is left for future work.

Definition 8. Let M and M' be two sets of literals. The *sd-distance* (sd="symmetric difference") $d_{sd}(M, M')$ between M and M' is defined via $d_{sd}(M, M') = |(M \cup M') \setminus (M \cap M')|$. If $\mathcal{M}$ is a set of sets of literals, we let

$$d_{sd}(M, \mathcal{M}) = \min_{M' \in \mathcal{M}} d(M, M').$$

It should be obvious that d_{sd} is indeed a distance function on sets according to Definition 7.

Now we can consider measures of the following kind.

Definition 9. Let d be a distance measure. Define $\mathcal{I}_d : \mathcal{P} \to \mathbb{R}^{\infty}_{\geq 0}$ via

$$\mathcal{I}_d(P) = \min_{\substack{M \subseteq \mathcal{L} \\ M \text{ consistent}}} \{d(M, M') \mid M' \in \text{Ans}_{Con}(P^M)\}$$

for all $P \in \mathcal{P}$ with $\min \emptyset = \infty$.

Note that we only allow consistent sets of literals because we want to measure the distance between a potential consistent answer set M of P and the consistent models of P^M. In the following, we abbreviate the inconsistency measure $\mathcal{I}_{d_{sd}}$ simply by $\mathcal{I}_{sd}$ and focus on this instance.

Example 3. Consider the simple case

$$P_3 : \qquad b \leftarrow \text{not } c. \qquad c \leftarrow \text{not } d. \qquad d \leftarrow \text{not } b.$$

We formally show that $\mathcal{I}_{sd}(P_3) = 1$. As the program is inconsistent, it is quite easy to see that $\mathcal{I}_{sd}(P_3) \geq 1$ holds. Now let $M = \{b\}$. Then the reduct is given via

$$P_3^M : \qquad b. \qquad c.$$

with minimal model $\{b, c\}$. We thus found a set M with

$$\exists M' \in \text{Ans}_{Con}(P_3^M) : \; d_{sd}(M, M') = 1.$$

Thus, $\mathcal{I}_{sd}(P_3) \leq 1$. For the program

$$P_2 : \qquad a \vee \neg a. \qquad \neg a \leftarrow a. \qquad a \leftarrow \neg a.$$

we see that $\mathcal{I}_{sd}(P_2) = \infty$ because there is no set M of literals such that P_2^M has a consistent answer set.

Remark 1. Note that $\mathcal{I}_{sd}(P) \in \{0, \infty\}$ if P is a classical program.

Another approach that is based on the semantics of a program P rather than the syntax is considering all answer sets of P and measuring the distance to a consistent one. Again, one could do this with arbitrary distances d. However, we will again only look at the symmetric difference d_{sd}. Given an inconsistent set M of literals, the minimal distance $d_{sd}(M, M')$ between M and a consistent set M' is simply the number of complementary literals in M. Let $\mathbb{N}_0$ denote the set of natural numbers including zero.

Definition 10. A set M of literals is called *k-inconsistent*, $k \in \mathbb{N}_0$, if there are exactly k atoms a such that $a \in M$ and $\neg a \in M$.

Further, we have to take into account that a program might be inconsistent due to having no answer set. We assign ∞ to such programs as they are a special case for this measure.

Definition 11. Define $\mathcal{I}_{\#} : \mathcal{P} \to \mathbb{R}^{\infty}_{\geq 0}$ via

$$\mathcal{I}_{\#}(P) = \min_{M \in \mathrm{Ans}(P)} \{k \mid M \text{ is } k\text{-inconsistent}\}$$

with $\min \emptyset = \infty$.

Example 4. Since

$$P_3 : \qquad b \leftarrow \text{not } c. \qquad c \leftarrow \text{not } d. \qquad d \leftarrow \text{not } b.$$

has no answer set, $\mathcal{I}_{\#}(P_3) = \infty$. For the program

$$P_2 : \qquad a \vee \neg a. \qquad \neg a \leftarrow a. \qquad a \leftarrow \neg a.$$

we obtain $\mathcal{I}_{\#}(P_2) = 1$ due to the inconsistent answer set $M = \{a,\ \neg a.\}$.

This concludes our discussion on distance-based measures.

3.3 Modification-based Measures

Our next measure $\mathcal{I}_{\pm}$ aims at measuring the effort needed to turn an inconsistent program into a consistent one. More specifically, it quantifies the number of modifications in terms of deleting and adding rules, necessary in order to restore consistency. Deleting certain rules can surely be sufficient to prevent P from entailing contradictions, but as already pointed out before, adding rules can also resolve inconsistency.

Definition 12. Define $\mathcal{I}_{\pm} : \mathcal{P} \to \mathbb{R}^{\infty}_{\geq 0}$ via

$$\mathcal{I}_{\pm}(P) = \min\{|A| + |D| \mid A, D \in \mathcal{P} \text{ such that } (P \cup A) \setminus D \text{ is consistent}\}$$

for all $P \in \mathcal{P}$.

Example 5. Let us consider our examples from above again. Since deleting any rule of

$$P_2 : \qquad a \vee \neg a. \qquad \neg a \leftarrow a. \qquad a \leftarrow \neg a.$$

renders the program consistent, $\mathcal{I}_{\pm}(P_2) = 1$. The same is true for

$$P_3'' : \qquad b \leftarrow \text{not } c. \qquad c \leftarrow \text{not } d. \qquad d \leftarrow \text{not } b.$$
$$d \leftarrow \text{not } e. \qquad e.$$

In the latter case, however, one could also *add* rules. For example, $P_3'' \cup \{d.\}$ is consistent.

The definition of $\mathcal{I}_\pm$ allows the addition of any rule in order to restore consistency. But in fact, it is sufficient to only consider addition of facts instead of general rules. First, we show that adding rules with disjunction in the head is not necessary, which is intuitive since ASP requires minimality anyway.

Proposition 1. *Let P be an inconsistent program. If r is a rule such that $P \cup \{r\}$ is consistent, then there is a literal $a \in head(r)$ such that $P \cup \{a \leftarrow body(r).\}$ is consistent.*

Proof. Let M be a consistent answer set of $P \cup \{r\}$. P being inconsistent implies that M is not an answer set of P. Thus, $M \models body(\{r\}^M)$ and $\{r\}^M \neq \emptyset$ because otherwise one could delete the rule while maintaining M as answer set. It follows that $head(r) \cap M \neq \emptyset$ since M is a model of $(P \cup \{r\})^M$. Let $a \in head(r) \cap M$. We show that M is an answer set of $P \cup \{a \leftarrow body(r).\}$ as well. By definition, M is a minimal model of $(P \cup \{r\})^M$. Since $a \in M$, M is a model of $(P \cup \{a \leftarrow body(r).\})^M$ as well. Now assume M is not a minimal model and let $M' \subsetneq M$ be a model of $(P \cup \{a \leftarrow body(r).\})^M$. Due to $a \in head(r)$ this implies that M' is a model of $(P \cup \{r\})^M$, too. Since M was assumed to be an answer set of $(P \cup \{r\})^M$, this yields a contradiction. □

Now we are ready to show that facts are sufficient.

Proposition 2. *Let P be an inconsistent program. If r is a rule such that $P \cup \{r\}$ is consistent, then there is a literal $a \in head(r)$ such that $P \cup \{a.\}$ is also consistent.*

Proof. Using Proposition 1, we assume that $head(r)$ contains only one literal a. As in the proof of Proposition 1, we see that $M \models body(\{r\}^M)$ and $\{r\}^M \neq \emptyset$. Since M is a model of P^M, we obtain $a \in M$. However, this means M is an answer set of $P \cup \{a.\}$. □

We now consider a simplified version of $\mathcal{I}_\pm$ that focuses entirely on additions of rules to restore consistency. This measure is adequate whenever the information in the given program is considered highly reliable and when it is thus more likely that information was forgotten rather than represented incorrectly. Adding additional assumptions seems to be a reasonable solution to resolve all the inconsistencies in such situations. This motivates the measure $\mathcal{I}_+$ which applies this solution.

Definition 13. Let $\mathcal{I}_+ : \mathcal{P} \to \mathbb{R}^\infty_{\geq 0}$ be the measure given via

$$\mathcal{I}_+(P) = \min\{|A| \mid A \in \mathcal{P} \text{ such that } P \cup A \text{ is consistent}\}$$

with $\min \emptyset = \infty$.

The case $\mathcal{I}_+(P) = \infty$ occurs whenever adding rules cannot resolve inconsistency, e. g., if a program contains two contradicting facts. Note that Proposition 2 applies to $\mathcal{I}_+$ as it does to $\mathcal{I}_\pm$. Hence, we can also w. l. o. g. assume the set A in the definition of $\mathcal{I}_+$ to be a set of facts.

Example 6. The program

$$P_2: \qquad a \vee \neg a. \qquad \neg a \leftarrow a. \qquad a \leftarrow \neg a.$$

cannot be repaired by adding rules. Hence, $\mathcal{I}_+(P_2) = \infty$. As argued before, the inconsistency of

$$\begin{aligned} P_3'': \qquad & b \leftarrow \text{not } c. \qquad c \leftarrow \text{not } d. \qquad d \leftarrow \text{not } b. \\ & d \leftarrow \text{not } e. \qquad e. \end{aligned}$$

can be resolved by adding e. g. "d.". Thus, $\mathcal{I}_+(P_3'') = 1$.

The following observation is obvious.

Proposition 3. *For any program $P \in \mathcal{P}$, $\mathcal{I}_\pm(P) \leq \mathcal{I}_+(P)$.*

Remark 2. If P is an inconsistent classical program, then adding rules will never resolve inconsistency. Hence $\mathcal{I}_+(P) \in \{0, \infty\}$ if P is a classical program.

Of course, it is also possible to focus entirely on deletions of rules. Since the empty program is obviously consistent, it is always possible to restore consistency via deletions alone. Deletions are the obvious choice when modeling errors may have occurred.

Definition 14. Let $\mathcal{I}_- : \mathcal{P} \to \mathbb{R}^\infty_{\geq 0}$ be the measure given via

$$\mathcal{I}_-(P) = \min\{|D| \mid D \in \mathcal{P} \text{ such that } P \setminus D \text{ is consistent}\}$$

for any $P \in \mathcal{P}$.

The following observations also follow directly by definition.

Proposition 4. *For any $P \in \mathcal{P}$, $\mathcal{I}_\pm(P) \leq \mathcal{I}_-(P)$. If P is a classical program, $\mathcal{I}_\pm(P) = \mathcal{I}_-(P)$.*

4 Rationality Postulates for Inconsistency Measures

Research in inconsistency measurement is driven by *rationality postulates*, i. e., desirable properties that should hold for concrete approaches. There is a growing number of rationality postulates for inconsistency measurement but not every postulate is generally accepted, see [5] for a recent discussion on this topic.

In the following, we revisit a selection of the most popular postulates—see [45] for a recent survey—and phrase them within our context of logic programs.

Let $\mathcal{I} : \mathcal{P} \to \mathbb{R}^{\infty}_{\geq 0}$ be some inconsistency measure and $P, P' \in \mathcal{P}$ some disjunctive logic programs. The most central property of any inconsistency measure is that it is able to distinguish consistency from inconsistency.

Consistency P is consistent iff $\mathcal{I}(P) = 0$.

The above postulate establishes that 0 is the minimal inconsistency value and that it is reserved for consistent programs.

We have already mentioned *Monotonicity* as a desirable property for inconsistency measures in monotonic logics in the introduction.

Monotonicity $\mathcal{I}(P) \leq \mathcal{I}(P')$ whenever $P \subseteq P'$.

Satisfaction of this postulate is generally *not* desirable for ASP, as we discussed before.

In the rest of this section we will propose new postulates for logic programs. We first discuss weaker variants of monotonicity in Section 4.1 and then discuss some further postulates in Section 4.2.

4.1 Weakening Monotonicity

Although monotonicity as a general property is undesired, as we have seen, we still wish to require some form of monotonicity in special cases. First, if a program P does not contain any default negation, no additional information can resolve any conflicts in P.

CLP-Monotonicity If P is a classical logic program and P' an arbitrary one, then $\mathcal{I}(P) \leq \mathcal{I}(P \cup P')$.

In the above postulate, CLP stands for "classical logic program". We can further elaborate on the idea of *CLP-Monotonicity*: Whenever P' has no influence on P, then $\mathcal{I}(P) \leq \mathcal{I}(P \cup P')$ should hold as well. To make this precise, we need the notion of the dependency graph of a program [2].

Definition 15. Let P be an extended logic program. The *dependency graph* D_P of the program P is a labeled directed graph having $\mathcal{L}(P)$ as vertices and there is an edge (l_i, l_j, s) iff P contains a rule r such that $l_j \in head(r)$ and $l_i \in pos(r) \cup neg(r)$. The label $s \in \{+, -\}$ indicates whether $l_i \in pos(r)$ or $l_i \in neg(r)$. For any $l \in \mathcal{L}(P)$, let $\mathsf{Path}(P, l)$ be the set of all literals l' such that there is a directed path (with any labels) from l to l' in D_P (including l itself). For a set M of literals, let

$$\mathsf{Path}(P, M) = \bigcup_{l \in M} \mathsf{Path}(P, l).$$

Example 7. Consider the union of the programs considered in Example 1 except the fact "e." i. e., the program P_4 given as follows:

$$
\begin{array}{lccc}
P_4: & a \vee \neg a. & \neg a \leftarrow a. & a \leftarrow \neg a.. \\
& b \leftarrow \text{not } c. & c \leftarrow \text{not } d. & d \leftarrow \text{not } b. \\
& d \leftarrow \text{not } e. &
\end{array}
$$

The dependency graph of P_4 is depicted in Figure 1. For example, $\mathsf{Path}(P_4, b) = \{b,\ c,\ d\}$.

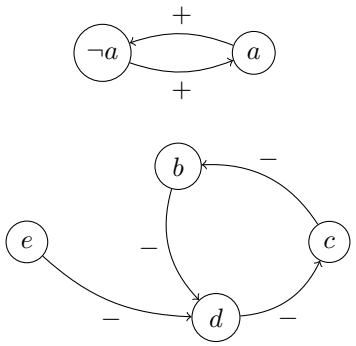

Figure 1: Dependency graph of P_4

Now we are ready to describe the notion of *splitting* logic programs [36].

Definition 16. Let P be an extended logic program. A set U of literals is called a *splitting set* for P, if $head(r) \cap U \neq \emptyset$ implies $\mathcal{L}(r) \subseteq U$ for any rule $r \in P$. For a splitting set U, let $bot_U(P)$ be the set of all rules $r \in P$ with $head(r) \subseteq U$. This set of rules is called the *bottom part* of P with respect to U.

In other words, if l is a literal that is not contained in U, i. e., $l \in \mathcal{L}(P) \setminus U$, then it cannot be in the head of any rule in $bot_U(P)$. For the dependency graph D_P, this means that while there could be a path from a literal $l' \in U$ to l, the converse is not true. Splitting is used, for example, to effectively compute answer sets because it allows handling $bot_U(P)$ without taking the rest of the program into account, see [36] for more details. However, since splitting is generally useful to examine the structure of a program, we are also interested in this notion here.

Example 8. Consider program P_4 again.

$$
\begin{array}{lllll}
P_4: & a \vee \neg a. & \neg a \leftarrow a. & a \leftarrow \neg a. \\
& b \leftarrow \text{not } c. & c \leftarrow \text{not } d. & d \leftarrow \text{not } b. \\
& d \leftarrow \text{not } e.
\end{array}
$$

For the splitting set $U = \{a, \neg a, e\}$, $bot_U(P_4)$ is the program

$$
\begin{array}{lllll}
bot_U(P_4): & a \vee \neg a. & \neg a \leftarrow a. & a \leftarrow \neg a.
\end{array}
$$

The definition of a splitting set ensures that the set of nodes that corresponds to $U = \{a, \neg a, e\}$ has no in-going edge from outside in the dependency graph, cf. Figure 1.

Theorem 1 ([36])**.** *Let U be a splitting set for a program P. Every answer set M of P is of the form $M = X \cup Y$ with an answer set X of $bot_U(P)$ and a set Y of literals.*

Corollary 1. *Let U be a splitting set of P. If P is consistent, then so is $bot_U(P)$.*

Proof. Let U be a splitting set of P and M a consistent answer set. Due to Theorem 1, there is a subset X of M such that X is an answer set of $bot_U(P)$. M being consistent implies X is consistent. Thus, $bot_U(P)$ is consistent. □

Example 9. We continue Example 8. The bottom part $bot_U(P_4)$ has one answer set, namely $\{a, \neg a\}$. Due to Theorem 1, all answer sets of P_4 contain both a and $\neg a$. Indeed, P_4 has the single answer set $\{a, \neg a, b, d, \}$. In particular, it is impossible to resolve the inconsistency of the program without changing the bottom part.

Intuitively, if $bot_U(P)$ is inconsistent, changing the rest of the program will not remove the reason why the bottom part is inconsistent; it is imposed on P. So, it is reasonable to assume that P is at least as inconsistent as $bot_U(P)$.

Split-Monotonicity If U is a splitting set for P, then $\mathcal{I}(bot_U(P)) \leq \mathcal{I}(P)$.

In the above notions of monotonicity, we always require properties ensuring that, once a program entails a contradiction, the added rules do not fix this. In the case of *CLP-Monotonicity*, this is done by requiring P to be a classical program and in the presence of a splitting set U, this property is inherent since the bottom part is just independent from the remainder of the program. However, we also achieve this goal if we "avoid" non-monotonicity: Adding a rule r to a program P will not resolve inconsistency as long as we make sure that the rule is not "involved in the non-monotonicity" of the program. This is the case if there are no paths from any literal of the head of r to default-negated literals. This ensures that the rule is meaningless for the derivation of such literals. To motivate this postulate, we make the following observations.

Lemma 1. *Let P be a disjunctive logic program and $r \notin P$ a rule with*

$$\mathsf{Path}(P \cup \{r\}, head(r)) \cap neg(P \cup \{r\}) = \emptyset. \tag{2}$$

Then, for all answer sets M^* of $P \cup \{r\}$, there is an answer set M of P with $M \subseteq M^*$.

Proof. For notational convenience, we let

$$P^* = P \cup \{r\}.$$

Let M^* be an answer set of P^*, i. e., $M^* \in \mathrm{Ans}_{Con}((P^*)^{M^*})$. The outline of the proof is as follows: From M^* we find a minimal model M of P^{M^*} with $M \subseteq M^*$. Using (2), we will see that $P^{M^*} = P^M$ and hence, M being a minimal model of P^{M^*} implies it is a minimal model of P^M as well. However, this means that M is an answer set of P.

So let M^* be a minimal model of $(P^*)^{M^*}$. Of course, M^* is a model of P^{M^*} as well, but it might not be minimal. Since P^{M^*} is a classical program, it has a minimal model. So, we can remove literals from M^* until we obtain a minimal model $M \subseteq M^*$ from P^{M^*}. We now examine the set $X = M^* \setminus M$. We assume $X \neq \emptyset$, i. e., M is a proper subset of M^*. Otherwise, the claim follows trivially. Hence, M is not a model of $(P^*)^{M^*}$, because M^* was assumed to be minimal. However, the only rule in $(P^*)^{M^*}$ that is not contained in P^{M^*} is $\{r\}^{M^*}$. Since M is a model of P^{M^*}, this means that M does not satisfy $\{r\}^{M^*}$.

As a minimal model of $(P^*)^{M^*}$, M^* satisfies $\{r\}^{M^*}$. Thus, we can find a literal $l_1 \in X = M^* \setminus M$ with $l_1 \in head(r)$. Now, $M_1 = M \cup \{l_1\}$ satisfies $\{r\}^{M^*}$. Clearly, $M_1 \subseteq M^*$. If $M_1 = M^*$, then we found that l_1 is the only literal that we need to add, i. e., that it is the only literal in $X = M^* \setminus M$. Otherwise, M_1 cannot be a model of $(P^*)^{M^*}$, because M^* was assumed to be minimal. In this case, we find a rule that is not satisfied and continue as above to obtain a set $M_2 = M_1 \cup \{l_2\} \subseteq M^*$. Since M^* is a model of $(P^*)^{M^*}$, we will never encounter a situation, where at some point M_i cannot be augmented with an additional literal to satisfy an unsatisfied rule. Since M^* is minimal, the procedure will not stop until we constructed M^*, i. e., found an n such that $M_n = M^*$. In particular, this means that we have to augment M with all literals in X.

Recall that the first literal we added, l_1, was contained in $head(r)$. Since M is a model of P^{M^*}, M satisfies all rules in P^{M^*}. Thus, the fact that we might have to add further literals stems from augmenting M with l_1. Using this observation, one can easily inductively verify that all literals in $X = M^* \setminus M$, i. e., all literals we added in the procedure described above, are contained in $\mathsf{Path}(P \cup \{r\}, head(r))$.

Now, we are ready to show that $P^{M^*} = P^M$. As pointed out, M^* is of the form

$$M^* = M \cup X \tag{3}$$

with

$$X \subseteq \mathsf{Path}(P^*, head(r)). \tag{4}$$

Now, (4) and (2) imply

$$X \cap neg(P^*) \subseteq \mathsf{Path}(P^*, head(r)) \cap neg(P^*) = \emptyset.$$

Since the construction of the reduct depends on literals in $neg(P)$ only, the literals in X can be ignored and thus we obtain

$$P^{M^*} = P^{M \cup X} = P^M. \tag{5}$$

As already mentioned, this implies that M is a minimal model of P^M as well and thus an answer set of P. □

In particular, this means that moving from $P \cup \{r\}$ to P cannot introduce inconsistency.

Corollary 2. *Let P be a disjunctive logic program and $r \notin P$ a rule with*

$$\mathit{Path}(P \cup \{r\}, head(r)) \cap neg(P \cup \{r\}) = \emptyset.$$

If $P \cup \{r\}$ is consistent, then so is P.

Proof. Let M^* be a consistent answer set of $P \cup \{r\}$. Due to Lemma 1, there is a subset $M \subseteq M^*$ such that M is an answer set of P. M^* being consistent implies M is consistent. So, P has a consistent answer set. □

Example 10. Consider again the inconsistent program P_3''.

$$\begin{array}{llll} P_3'': & b \leftarrow \text{not } c. & c \leftarrow \text{not } d. & d \leftarrow \text{not } b. \\ & d \leftarrow \text{not } e. & e. & \end{array}$$

If we add the fact "$r = d$.", the program becomes consistent, having $\{b, d\}$ as the only answer set. However, we have $d \in neg(P)$ and in particular,

$$\mathsf{Path}(P_3'' \cup \{r\}, head(r)) \cap neg(P_3'' \cup \{r\}) \neq \emptyset$$

and thus, this example does not satisfy the premise of Corollary 2.

Due to the above considerations, we deem the following postulate as desirable (I="Independence").

I-Monotonicity If r is a rule with $\mathsf{Path}(P \cup \{r\}, head(r)) \cap neg(P \cup \{r\}) = \emptyset$, then $\mathcal{I}(P) \leq \mathcal{I}(P \cup \{r\})$.

So far, we considered situations where augmenting a program P with rules cannot prevent the entailment of contradictions. However, there is another situation where a rule r should never decrease the severity of inconsistency—namely, if r is a constraint:

Definition 17. Let P be a disjunctive logic program. A rule of the form

$$a \leftarrow l_1, \ldots, l_k, \text{ not } l_{k+1}, \ldots, \text{not } l_m, \text{ not } a. \tag{6}$$

where a is an atom which does not occur elsewhere in the program is called a *constraint*.

Intuitively, a constraint $r \in P$ of the form (6) ensures that a set M of literals cannot be an answer set of P if M contains the literals $\{l_1, \ldots, l_k\}$, but none of $l_{k+1}, \ldots, l_m$. Hence, constraints reduce the amount of answer sets of a program. It is straightforward to see that the inconsistency of P implies the inconsistency of $P \cup \{r\}$ for a constraint r.

This motivates the following postulate (Con="Constraint").

Con-Monotonicity If r is a constraint, then $\mathcal{I}(P) \leq \mathcal{I}(P \cup \{r\})$ for any $P \in \mathcal{P}$.

This concludes our discussion on weaker versions of monotonicity.

4.2 Further Postulates

We will now discuss postulates which consider cases where the inconsistency value should *not* change. For this reason, we consider established notions of equivalence for logic programs, cf. [35, 49].

Definition 18. Two logic programs P, P' are *equivalent*, denoted by $P \equiv P'$, if $\text{Ans}(P) = \text{Ans}(P')$. They are *strongly equivalent*, denoted by $P \equiv_s P'$, if $\text{Ans}(P \cup H) = \text{Ans}(P' \cup H)$ for every $H \in \mathcal{P}$.

The notion of equivalence is only useful if we are interested in the answer sets of a given program. Two programs can be equivalent while encoding different information, e. g.,

$$P: \begin{array}{l} b \leftarrow \text{not } a. \\ \neg b \leftarrow \text{not } a. \end{array} \qquad\qquad P': \begin{array}{l} b. \\ \neg b. \end{array}$$

Furthermore, the above programs behave differently if we consider them as parts of a larger program. Even though this shows that this notion of equivalence is weak, respecting equivalent programs still seems to be a desirable property for inconsistency measures that are determined by the answer sets of a program rather than other aspects.

E-Indifference If $P \equiv P'$, then $\mathcal{I}(P) = \mathcal{I}(P')$.

The analogous postulate for strongly equivalent programs is obviously even more desirable.

SE-Indifference If $P \equiv_s P'$, then $\mathcal{I}(P) = \mathcal{I}(P')$.

Due to the definition of $\equiv_s$, the postulate *Exchange* from [5], which is advocated by Besnard as desirable for propositional logics, seems reasonable for strongly equivalent programs P and P', too.

Exchange If $P \equiv_s P'$, then for all $H \in \mathcal{P}$, $\mathcal{I}(P \cup H) = \mathcal{I}(P' \cup H)$.

However, *Exchange* and *SE-Indifference* coincide.

Proposition 5. *Let $\mathcal{I}$ be an inconsistency measure. If $\mathcal{I}$ satisfies* E-Indifference*, then it satisfies* SE-Indifference *as well. Furthermore, it satisfies* SE-Indifference *if and only if it satisfies* Exchange*.*

Proof. The first statement is clear. The satisfaction of *Exchange* implies the satisfaction of *SE-Indifference*, because we can choose $H = \emptyset$. The converse holds since $P \cup H \equiv_s P' \cup H$ for any program H. □

Our final two postulates are concerned with the language used in a program. Intuitively, parts of a program which do not share any vocabulary elements with the rest of the program should be assessed separately with respect to inconsistency.

Language Separability If $\mathcal{A}(P) \cap \mathcal{A}(P') = \emptyset$, then $\mathcal{I}(P \cup P') = \mathcal{I}(P) + \mathcal{I}(P')$.

Example 11. Consider again the following programs from Example 1

$$
\begin{array}{lrrr}
P_2: & a \vee \neg a. & \neg a \leftarrow a. & a \leftarrow \neg a.. \\
P_3'': & b \leftarrow \text{not } c. & c \leftarrow \text{not } d. & d \leftarrow \text{not } b. \\
 & d \leftarrow \text{not } e. & e. &
\end{array}
$$

As they do not share any atoms, their conflicts should be considered independent and hence,

$$\mathcal{I}(P_2 \cup P_3'') = \mathcal{I}(P_2) + \mathcal{I}(P_3'')$$

should hold.

Another approach builds on the notion of *safe* formulas [44]. A consistent formula α is *safe* with respect to a set $\mathcal{K}$ of propositional formulas if α and $\mathcal{K}$ do not share any atoms. So, adding α to $\mathcal{K}$ will never introduce inconsistency. The corresponding postulate *safe-formula independence* requires $\mathcal{I}(\mathcal{K}) = \mathcal{I}(\mathcal{K} \cup \{\alpha\})$ whenever α is safe with respect to $\mathcal{K}$.

Due to the directedness of rules we can be somewhat more liberal in defining a corresponding notion of safeness for the ASP-setting and do not need to require completely disjoint languages for r and P. There are two different ways in which a single rule r can have an effect on the consistency of a program P. First of all, the literals in the head of r may interact with the the program and

thus introduce or eliminate inconsistency. To make sure this kind of interaction cannot happen, we have to require that the atoms in the head of r are disjoint from the atoms appearing in P. But there is another, more indirect way in which a new rule can cause contradiction, namely if r is self-contradictory, in the sense that a literal is derivable if and only if it is not derivable. The simplest example is the rule "$a \leftarrow$ not a." which leads to the non-existence of answer sets if added to a program P whenever atom a does not appear in P. To eliminate the possibility of this phenomenon we require that the literals in the head of r not occur default-negated in the body of r. This leads to the following definition.

Definition 19. Let P be a disjunctive logic program. A rule r is called *safe* with respect to P if $\mathcal{A}(head(r)) \cap \mathcal{A}(P) = \emptyset$ and $head(r) \cap neg(r) = \emptyset$.

Our discussion motivates the following postulate.

Safe-Rule Independence If P is a logic program and r safe with respect to P, then $\mathcal{I}(P) = \mathcal{I}(P \cup \{r\})$.

Remark 3. Consider a disjunctive logic program P and a rule r that is safe with respect to P. We can view $U = \mathcal{L}(P)$ as a splitting set of $P \cup \{r\}$ and obtain P as a bottom part. Then, for any measure $\mathcal{I}$ that satisfies Split-Monotonicity, we have $\mathcal{I}(P) \leq \mathcal{I}(P \cup \{r\})$. The same holds for any measure $\mathcal{I}$ satisfying I-Monotonicity, since $\mathsf{Path}(P \cup \{r\}, head(r)) \cap neg(P \cup \{r\}) = \emptyset$ is clear for a safe rule r. Neither Split-Monotonicity nor I-Monotonicity implies Safe-Rule Independence, though, since we do not obtain $\mathcal{I}(P) \geq \mathcal{I}(P \cup \{r\})$.

5 Compliance with Rationality Postulates

Table 1 gives an overview of the compliance of our measures with respect to the rationality postulates from Section 4 and thus summarizes Propositions 6-12 and Examples 12-14 below. Note that, naturally, none of our measures satisfies the ordinary *Monotonicity* postulate which is also not desired for ASP, cf. Section 1.

Proposition 6. $\mathcal{I}_{01}$ *satisfies* Consistency, CLP-Monotonicity, Split-Monotonicity, I-Monotonicity, Con-Monotonicity, E-Indifference, SE-Indifference, Exchange *and* Safe-Rule Independence

Proof. Split-Monotonicity follows from Corollary 1 and *I-Monotonicity* from Corollary 2. The rest is clear. □

Naturally, $\mathcal{I}_{01}$ does not satisfy *Language Separability* since $\mathcal{I}_{01}(P) \in \{0, 1\}$ for any program $P \in \mathcal{P}$.

	$\mathcal{I}_{01}$	$\mathcal{I}_{\pm}$	$\mathcal{I}_{+}$	$\mathcal{I}_{-}$	$\mathcal{I}_{\mathsf{MSI}}$	$\mathcal{I}_{\#}$	$\mathcal{I}_{sd}$
Consistency	✓	✓	✓	✓	✓	✓	✓
Monotonicity	✗	✗	✗	✗	✗	✗	✗
CLP-Monotonicity	✓	✓	✓	✓	✓	✓	✓
Split-Monotonicity	✓	✓	✓	✓	✓	✓	✓
I-Monotonicity	✓	✓	✓	✓	✓	✓	✓
Con-Monotonicity	✓	✓	✓	✓	✓	✓	✓
E-Indifference	✓	✗	✗	✗	✗	✓	✗
SE-Indifference	✓	✗	✓	✗	✗	✓	✗
Exchange	✓	✗	✓	✗	✗	✓	✗
Language Separability	✗	✓	✓	✓	✓	✓	✓
Safe-Rule Independence	✓	✓	✓	✓	✓	✓	✓

Table 1: Compliance of inconsistency measures with respect to our rationality postulates

Proposition 7. *$\mathcal{I}_{\pm}$ satisfies* Consistency, CLP-Monotonicity, Split-Monotonicity, I-Monotonicity, Con-Monotonicity, Language Separability *and* Safe-Rule Independence.

Proof. **Consistency** Clear from the definition of $\mathcal{I}_{\pm}$.

CLP-Monotonicity Let P be a classical program and P' an arbitrary one. Assume $\mathcal{I}_{\pm}(P \cup P') = k$. Let $(P \cup P' \cup A) \setminus D$ be consistent with $|A| + |D| = k$. Since P is a classical logic program, $P \setminus D$ must be consistent as well since adding rules cannot restore consistency. Hence, $\mathcal{I}_{\pm}(P) \leq |D| \leq |A| + |D| = k$.

Split-Monotonicity Let P be a program and U a splitting set. Let $\mathcal{I}_{\pm}(P) = k$ and let $(P \cup A) \setminus D$ be consistent with $|A| + |D| = k$. We use Proposition 2 to assume that A is a set of facts. Thus, $(P \cup A) \setminus D$ contains no additional edges in the dependency graph compared to P which implies that U is a splitting set of $(P \cup A) \setminus D$ as well. In particular, if we let A_U be the subset of A such that $r \in A_U$ if and only if $head(r) \subseteq U$, then $(bot_U(P)) \cup A_U) \setminus D$ is the corresponding bottom program. Now let M be a consistent answer set of $(P \cup A) \setminus D$. Due to Theorem 1, there is a subset $X \subseteq M$ such that X is an answer set of $(bot_U(P)) \cup A_U) \setminus D$. As a subset of the consistent set M of literals, X is consistent. Therefore, $(bot_U(P)) \cup A_U) \setminus D$ is consistent. Hence,

$$\mathcal{I}_{\pm}(bot_U(P)) \leq |D| + |A_U| \leq |D| + |A| = k.$$

I-Monotonicity Let $\mathcal{I}_{\pm}(P \cup \{r\}) = k$. Let $(P \cup \{r\} \cup A) \setminus D$ be consistent with $|A| + |D| = k$. Let M^* be a consistent answer set of $(P \cup \{r\} \cup A) \setminus D$.

Using Proposition 2, we assume that A is a set of facts. So, since

$$\mathsf{Path}(P \cup \{r\}, head(r)) \cap neg(P \cup \{r\}) = \emptyset,$$

we also obtain

$$\mathsf{Path}(((P \cup A) \setminus D) \cup \{r\}, head(r)) \cap neg(P \cup \{r\}) = \emptyset,$$

because adding facts (and deleting rules) does not extend the dependency graph. Furthermore, we have

$$neg(((P \cup A) \setminus D) \cup \{r\}) = neg((P \setminus D) \cup \{r\}) \subseteq neg(P \cup \{r\})$$

and hence,

$$\mathsf{Path}(((P \cup A) \setminus D) \cup \{r\}, head(r)) \cap neg(P \cup \{r\}) = \emptyset$$

also implies

$$\mathsf{Path}(((P \cup A) \setminus D) \cup \{r\}, head(r)) \cap neg(((P \cup A) \setminus D) \cup \{r\}) = \emptyset.$$

So, we can apply Lemma 1 to the program $(P \cup A) \setminus D$ and the additional rule r. Since M^* was assumed to be an answer set of $(P \cup \{r\} \cup A) \setminus D$, we obtain that $(P \cup A) \setminus D$ has an answer set M with $M \subseteq M^*$. With M^* being consistent, M is consistent, too. Likewise, $(P \cup A) \setminus D$ is consistent. Thus, $\mathcal{I}_{\pm}(P) \leq |A| + |D| = k = \mathcal{I}_{\pm}(P \cup \{r\})$.

Con-Monotonicity If r is a constraint and $(P \cup \{r\} \cup A) \setminus D$ is consistent, then $(P \cup A) \setminus D$ is consistent as well.

Language Separability This is clear since P and P' need to be considered independently.

Safe-Rule Independence That is clear, since no matter which facts A we add or rules D we delete, whether or not $(P \cup A) \setminus D$ is consistent will never depend on a safe rule r. The only exception would be adding a fact "$\neg a$." for an atom $a \in head(r)$ which, however, will never be beneficial to resolve the inconsistency of P anyway. □

It is clear that $\mathcal{I}_{\pm}$ does not satisfy the remaining postulates *(S)E-Indifference* and *Exchange* since the notion of (strong) equivalence does not take into account *how many* rules a program contains to ensure the entailment of certain atoms. The measure $\mathcal{I}_{\pm}$, however, is tailored to assess this. Consider, for example

Example 12.

$$\begin{array}{lllll} P: & a. & \neg a. & & \\ P': & a. & \neg a. & & \\ & b. & a \leftarrow b. & \neg a \leftarrow a. \end{array}$$

with $\mathcal{I}_{\pm}(P) = 1$ and $\mathcal{I}_{\pm}(P') = 2$

In contrast, the measure $\mathcal{I}_+$ satisfies *SE-Indifference* and *Exchange* as only adding rules is considered here. The rest is rather similar to $\mathcal{I}_{\pm}$.

Proposition 8. *$\mathcal{I}_+$ satisfies* Consistency, CLP-Monotonicity, Split-Monotonicity, I-Monotonicity, Con-Monotonicity, SE-Indifference, Exchange, Language Separability *and* Safe-Rule Independence.

Proof. **Consistency** This is clear.

CLP-Monotonicity If P is consistent, then $\mathcal{I}_+(P) = 0$. If P is inconsistent, then $\mathcal{I}_+(P) = \infty$ since adding rules cannot resolve inconsistency. For the same reason, P being inconsistent implies that $P \cup P'$ is inconsistent with $\mathcal{I}_+(P \cup P') = \infty$. Hence, in both cases, we obtain $\mathcal{I}_+(P) \leq \mathcal{I}_+(P \cup P')$.

Split-Monotonicity, **I-Monotonicity** and **Con-Monotonicity** Similar to $\mathcal{I}_{\pm}$.

SE-Indifference *Exchange* and Proposition 5.

Exchange Let $P \equiv_s P'$ and let $H \in \mathcal{P}$. Let $\mathcal{I}_+(P \cup H) = k$. This means that there is a set A of facts with $|A| = k$ such that $P \cup H \cup A$ is consistent. $P \equiv_s P'$ implies that $P' \cup H \cup A$ is consistent as well, yielding $\mathcal{I}_+(P' \cup H) \leq k = \mathcal{I}_+(P \cup H)$. Of course, we obtain $\mathcal{I}_+(P \cup H) \leq \mathcal{I}_+(P' \cup H)$ similarly.

Language Separability and **Safe-Rule Independence** Similar to $\mathcal{I}_{\pm}$. □

The postulate *E-Indifference* is not satisfied by $\mathcal{I}_+$.

Example 13. The programs

$$\begin{array}{lll} P: & b \leftarrow \text{not } a. & \qquad P': \ b. \\ & \neg b \leftarrow \text{not } a. & \qquad\quad \neg b. \end{array}$$

are equivalent. However, $\mathcal{I}_+(P) = 1$ and $\mathcal{I}_+(P') = \infty$.

Proposition 9. $\mathcal{I}_-$ *satisfies* Consistency, CLP-Monotonicity, Split-Monotonicity, I-Monotonicity, Con-Monotonicity, Language Separability *and* Safe-Rule Independence.

Proof. Similar to $\mathcal{I}_\pm$. □

For the same reason as $\mathcal{I}_\pm$, the measure $\mathcal{I}_-$ does not satisfy *(S)E-Indifference* and *Exchange*.

We now turn to the measure $\mathcal{I}_{\mathsf{MSI}}$.

Proposition 10. $\mathcal{I}_{\mathsf{MSI}}$ *satisfies* Consistency, CLP-Monotonicity, Split-Monotonicity, I-Monotonicity, Con-Monotonicity, Language Separability *and* Safe-Rule Independence.

Proof. **Consistency** As already pointed out in [11], $SI_{min}(P) = \emptyset$ iff P is consistent.

CLP-Monotonicity Let P be a classical program and P' an arbitrary one. If H is strongly P-inconsistent, then H is strongly $P \cup P'$-inconsistent as well, because conflicts within P cannot be resolved by adding rules. Similarly, one can see that the minimality of H is preserved. Hence, $H \in SI_{min}(P)$ implies $H \in SI_{min}(P \cup P')$. Hence, $SI_{min}(P) \subseteq SI_{min}(P \cup P')$ which implies $\mathcal{I}_{\mathsf{MSI}}(P) \leq \mathcal{I}_{\mathsf{MSI}}(P \cup P')$.

Split-Monotonicity Let P be a program and U a splitting set. By Theorem 1 (and similar considerations as above) we obtain $SI_{min}(bot_U(P)) \subseteq SI_{min}(P)$. The claim follows.

I-Monotonicity Similar, utilizing Lemma 1.

Con-Monotonicity Similar as constraints cannot restore consistency.

Language Separability and **Safe-Rule Independence** are clear. □

Regarding *(S)E-Indifference* and *Exchange*, the situation is similar as in the case of $\mathcal{I}_\pm$.

Proposition 11. $\mathcal{I}_{sd}$ *satisfies* Consistency, CLP-Monotonicity, Split-Monotonicity, I-Monotonicity, Con-Monotonicity, Language Separability *and* Safe-Rule Independence.

Proof. **Consistency** This is clear by definition.

CLP-Monotonicity If P is inconsistent, then it has only inconsistent answer sets. In this case, the reduct of $P \cup P'$ with respect to any set of literals has only inconsistent answer sets as well. Hence, $\mathcal{I}_{sd}(P \cup P') = \infty$. If P is consistent,

then $\mathcal{I}_{sd}(P) = 0$. In both cases, $\mathcal{I}_{sd}(P) \leq \mathcal{I}_{sd}(P \cup P')$ holds.

Split-Monotonicity Let P be a disjunctive logic program, U a splitting set and $bot_U(P)$ the corresponding bottom program. The case $\mathcal{I}_{sd}(P) = \infty$ is clear. We consider $\mathcal{I}_{sd}(P) = k < \infty$. Let M be a consistent set of literals with $d_{sd}(M, M') = k$ for an $M' \in \text{Ans}_{Con}(P^M)$. Note that U is also a splitting set of P^M with $(bot_U(P))^M$ as the corresponding bottom part. So, due to Theorem 1, M' is of the form $X' \cup Y'$ with

$$X' \in \text{Ans}_{Con}((bot_U(P))^M).$$

We can w. l. o. g. assume $X' \cap Y' = \emptyset$.
Now consider $X = M \cap U$ and $Y = M \setminus X$. Then, similarly to X' and Y' we have $X \cup Y = M$ and $X \cap Y = \emptyset$.
We obtain $(bot_U(P))^M = (bot_U(P))^X$ due to the construction of the reduct and U being a splitting set. Thus,

$$X' \in \text{Ans}_{Con}((bot_U(P))^X).$$

The last step argues that the constructed sets are disjoint: We have $X, X' \subseteq U$ and $Y \cap U = Y' \cap U = \emptyset$. So, $X \cap Y' = X' \cap Y = \emptyset$. Furthermore, $X \cap Y = X' \cap Y' = \emptyset$ was already mentioned. Thus, we can calculate

$$d_{sd}(X, X') \leq d_{sd}(X \cup Y, X' \cup Y') = d_{sd}(M, M') = k.$$

To summarize, we found two sets X, X' of literals with $X' \in \text{Ans}_{Con}((bot_U(P))^X)$ and $d_{sd}(X, X') \leq k$. Hence, $\mathcal{I}_{sd}(bot_U(P)) \leq k$.

I-Monotonicity Again, we only have to consider the case $\mathcal{I}_{sd}(P \cup \{r\}) = k < \infty$. Let M^* be a consistent set of literals such that $(M^*)' \in \text{Ans}_{Con}((P \cup \{r\})^{M^*})$ is consistent with $d_{sd}(M^*, (M^*)') = k$.
Consider $P^{(M^*)}$. As seen in the proof of Lemma 1, there is a set $M' \in \text{Ans}_{Con}(P^{M^*})$ of literals such that $(M^*)'$ is of the form $(M^*)' = M' \cup X'$ for a set X' of literals with $P^{(M^*)'} = P^{M' \cup X'} = P^{M'}$ (cf. (3) and (5)). Furthermore, M' and X' are disjoint.
Now consider $M = M^* \setminus X'$. We have $M' \in \text{Ans}_{Con}(P^{M^*}) = \text{Ans}_{Con}(P^{M^* \setminus X'})$ and

$$d_{sd}(M, M') = d_{sd}(M^* \setminus X', (M^*)' \setminus X') \leq d_{sd}(M^*, (M^*)') = k.$$

This implies $\mathcal{I}_{sd}(P) \leq k$.

Con-Monotonicity If $M' \in \text{Ans}_{Con}((P \cup \{r\})^M)$ with $d_{sd}(M, M') = k$, then we have $M' \in \text{Ans}_{Con}(P^M)$ as well and hence $\mathcal{I}_{sd}(P) \leq k$.

Language Separability and **Safe-Rule Independence** are clear. □

The following example shows that $\mathcal{I}_{sd}$ does not satisfy *SE-Indifference*. It makes use of the following observation: Even for two strongly equivalent logic programs P and P', the corresponding reducts P^M and P'^M with respect to to a given set M of literals might not be equivalent. It follows that $\mathcal{I}_{sd}$ does not satisfy *E-Indifference* or *Exchange*, either.

Example 14. Consider the following program P_5.

$$\begin{array}{lllll} P_5: & a. & b_1 \leftarrow \text{not } a. & & b_2 \leftarrow \text{not } a. \\ & & c_1 \leftarrow \text{not } b_1, \text{not } c_1. & & c_2 \leftarrow \text{not } b_2, \text{not } c_2. \end{array}$$

The program is inconsistent since b_1 and b_2 are not entailed. We obtain the best set of literals if we get rid of a (to be able to use the default "not a"). The corresponding reduct is

$$P_5^{\{b_1,\, b_2\}}: \qquad a. \qquad b_1. \qquad b_2.$$

with $d_{sd}(\{b_1,\, b_2\}, \text{Ans}_{Con}(P_5^{\{b_1,\, b_2\}})) = 1$. Since the program is inconsistent (i. e., $\mathcal{I}_{sd}(P_5) \geq 1$), $\mathcal{I}_{sd}(P_5) = 1$. Now consider the same program with two additional rules that prevent us from considering sets of literals without a.

$$\begin{array}{lllll} P_6: & a. & b_1 \leftarrow \text{not } a. & & b_2 \leftarrow \text{not } a. \\ & & c_1 \leftarrow \text{not } b_1, \text{not } c_1. & & c_2 \leftarrow \text{not } b_2, \text{not } c_2. \\ & & d \leftarrow \text{not } a. & & \neg d \leftarrow \text{not } a. \end{array}$$

For any set M of literals with $a \notin M$, the reduct P_6^M contains "d." and "$\neg d$.". Hence, we have to consider sets containing a and thus, the reduct will never contain "b_1." or "b_2.". Now one can check that $\mathcal{I}_{sd}(P_6) = 2$: For example, if we use the set $\{a,\, b_1\}$ we obtain the reduct

$$P_6^{\{a,\, b_1\}}: \qquad a. \qquad c_2.$$

with minimal model $\{a,\, c_2\}$, i. e., the distance is 2.

However, the programs P_5 and P_6 are strongly equivalent since the fact "a." renders the rules "$d \leftarrow \text{not } a$." and "$\neg d \leftarrow \text{not } a$." meaningless. Thus, *SE-Indifference* is not satisfied.

Proposition 12. *$\mathcal{I}_{\#}$ satisfies* Consistency, CLP-Monotonicity, Split-Monotonicity, I-Monotonicity, Con-Monotonicity, E-Indifference, SE-Indifference, Exchange, Language Separability *and* Safe-Rule Independence.

Proof. **Consistency** Clear due to the definition.

CLP-Monotonicity Clear due to the monotonicity of classical logic programs.

Split-Monotonicity Let P be a program and U a splitting set. Let $\tilde{X}$ be an answer set of $bot_U(P)$ with a minimal amount of complementary literals, say $2k$. Thus, $\mathcal{I}_{\#}(bot_U(P)) = k$. Now let M be an answer set of P. Due to Theorem 1, $X \subseteq M$ for an answer set X of $bot_U(P)$. Thus, M contains at least as many complementary literals as X, which itself contains at least as many as $\tilde{X}$. Hence, $\mathcal{I}_{\#}(P) \geq k$.

I-Monotonicity Let $\mathcal{I}_{\#}(P \cup \{r\}) = k^*$ and let M^* be a k^*-inconsistent answer set of $P \cup \{r\}$. By Lemma 1, there is an answer set M of P with $M \subseteq M^*$. So, if M is k-consistent, then $k \leq k^*$ and we obtain $\mathcal{I}_{\#}(P) \leq k \leq k^* = \mathcal{I}_{\#}(P \cup \{r\})$.

Con-Monotonicity If $\mathcal{I}_{\#}(P \cup \{r\}) = k$ and M is a k-inconsistent answer set of $P \cup \{r\}$, then M is a k-inconsistent answer set of P as well and hence, $\mathcal{I}_{\#}(P) \leq k$.

E-Indifference Clear due to the definition of $\mathcal{I}_{\#}$.

SE-Indifference *E-Indifference* and Proposition 5.

Exchange *E-Indifference* and Proposition 5.

Language Separability The number of complementary literals adds up.

Safe-Rule Independence This is clear. □

Our analysis shows that the measures investigated in this paper satisfy most of the postulates we discussed—with the exception of monotonicity, which, as explained, is not desirable in our context anyway. This provides our measures with some basic justification and shows that they behave in an expected way. The proposed postulates were adapted from existing ones for propositional logics and only modifications necessary to account for the non-monotonicity of ASP were made.

6 Computational Complexity

We now address the computational complexity of the measures we developed in this paper. After establishing some notation and making some general observations we investigate the complexity of our measures on disjunctive logic programs in Section 6.1. Afterwards, we have a look at some special cases. In particular, we discuss the complexity of disjunction-free programs in Section 6.2, stratified programs in Section 6.3, and classical disjunction-free programs in Section 6.4. The techniques we are going to use for our investigation

are not applicable for the measure $\mathcal{I}_{\mathsf{MSI}}$, though, and we leave a rigorous investigation of its computational complexity for future work.

Following [48], we consider the three decision problems $\textsc{Exact}_{\mathcal{I}}$, $\textsc{Upper}_{\mathcal{I}}$, $\textsc{Lower}_{\mathcal{I}}$, and the natural function problem $\textsc{Value}_{\mathcal{I}}$. Let $\mathcal{I}$ be an arbitrary inconsistency measure on logic programs. As we will investigate complexity with respect to different subclasses of logic programs, let $X \subseteq \mathcal{P}$ be some set of logic programs.

$\textsc{Exact}_{\mathcal{I}}^{X}$	**Input**:	$P \in X$, $x \in [0, \infty]$
	Output:	TRUE iff $\mathcal{I}(P) = x$
$\textsc{Upper}_{\mathcal{I}}^{X}$	**Input**:	$P \in X$, $x \in [0, \infty]$
	Output:	TRUE iff $\mathcal{I}(P) \leq x$
$\textsc{Lower}_{\mathcal{I}}^{X}$	**Input**:	$P \in X$, $x \in (0, \infty]$
	Output:	TRUE iff $\mathcal{I}(P) \geq x$
$\textsc{Value}_{\mathcal{I}}^{X}$	**Input**:	$P \in X$
	Output:	The value of $\mathcal{I}(P)$

We assume the reader to be familiar with the complexity classes P, NP and coNP. We make use of the polynomial hierarchy, defined using oracle machines as $\Sigma_0^p := \Pi_0^p := \mathsf{P}$ and $\Sigma_{i+1}^p := \mathsf{NP}^{\Sigma_i^p}$ and $\Pi_{i+1}^p := \mathsf{coNP}^{\Sigma_i^p}$ for $i \geq 0$. Here, $\mathcal{C}^{\mathcal{D}}$ is the class of decision problems solvable in $\mathcal{C}$ having access to an oracle for some problem that is complete in $\mathcal{D}$. The class D_i^p consists of all languages that are the intersection of a language in Σ_i^p and a language in Π_i^p. We also consider the functional complexity classes $\mathsf{FP}^{\mathsf{NP}[\log n]}$ and $\mathsf{FP}^{\Sigma_2^p[\log n]}$, i. e., classes that contain problems whose solution can be computed in P with access to a logarithmically bounded number of calls to an NP resp. Σ_2^p oracle.

In order to discuss computational complexity we need a measure of the size of the input to our problems. For that we use the following straightforward notion of the length of a logic program.

Definition 20. Let r be a rule of the form

$$l_0 \vee \ldots \vee l_k \leftarrow l_{k+1}, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n.$$

Then the length $\text{len}(r)$ of r is defined as $\text{len}(r) = n + 1$. For a program P, its length is defined via $\text{len}(P) = \sum_{r \in P} \text{len}(r)$.

The work [48] already established some relationships between the individual computational problems for the propositional case, in particular, results pertaining to the bounds for the complexity of other problems if the complexity of one problem is known. We will now extend these results to the case of logic programs. For that, we introduce a simple notion of *expressivity* for our measures, cf. [46].

Definition 21. Let $\mathcal{I}$ be an inconsistency measure. The *expressivity* $\mathcal{C}_\mathcal{I}$ of $\mathcal{I}$ is the function $\mathcal{C}_\mathcal{I} : \mathbb{N} \to 2^\mathbb{R}$ defined via $\mathcal{C}_\mathcal{I}(n) = \{\mathcal{I}(P) \mid \text{len}(P) \leq n\}$ for all $n \in \mathbb{N}$.

In other words, $\mathcal{C}_\mathcal{I}(n)$ is the set of values $\mathcal{I}$ can attain on logic programs of length n or smaller. For our measures we can see that the size of this set is bounded linearly by the length of the programs.

Lemma 2. *For* $\mathcal{I} \in \{\mathcal{I}_{01}, \mathcal{I}_\pm, \mathcal{I}_+, \mathcal{I}_-, \mathcal{I}_{sd}, \mathcal{I}_\#\}$, $\mathcal{C}_\mathcal{I}(n) \subseteq \{0, \ldots, n, \infty\}$.

Proof. For $\mathcal{I} \in \{\mathcal{I}_\pm, \mathcal{I}_-\}$, the claim follows from $|P| \leq |\text{len}(P)|$ and the observation that the empty program is consistent. Since we can consider adding facts only in the case of $\mathcal{I}_+$, at most $|\mathcal{L}(P)| \leq |\text{len}(P)|$ rules can be added. If adding rules cannot restore consistency, the value is ∞. Hence, $\mathcal{I}_+(P) \leq n+1$. The case $\mathcal{I} \in \{\mathcal{I}_{sd}, \mathcal{I}_\#\}$ is similar using $|\mathcal{L}(P)| \leq |\text{len}(P)|$. □

The above results allow us to adopt Lemma 2 from [48] which provides an upper bound for the complexity of $\textsc{Value}_\mathcal{I}^X$ given the complexity of $\textsc{Upper}_\mathcal{I}^X$. As there are only linearly many different values of $\mathcal{I}$ one can perform a binary search on these values in order to realize an algorithm for $\textsc{Value}_\mathcal{I}^X$ while using only logarithmically many calls to an algorithm for $\textsc{Upper}_\mathcal{I}^X$.

Lemma 3 (Lemma 2 in [48])**.** *Let* $\mathcal{I}$ *be an inconsistency measure,* $i > 0$ *an integer, and* $X \subseteq \mathcal{P}$*. If* $\textsc{Upper}_\mathcal{I}^X$ *is in* Σ_i^p *or* Π_i^p *and* $|\mathcal{C}_\mathcal{I}(n)| \in \mathcal{O}(n^k)$ *for some* $k \in \mathbb{N}$*, then* $\textsc{Value}_\mathcal{I}^X$ *is in* $\mathsf{FP}^{\Sigma_i^p[\log n]}$.

Naturally, we expect $\textsc{Upper}_\mathcal{I}^X$ and $\textsc{Lower}_\mathcal{I}^X$ to be complementary problems and $\textsc{Exact}_\mathcal{I}^X$ a combination of both. However, in order to see this, we adopt the following notion from [48].

Definition 22. An inconsistency measure $\mathcal{I}$ is called *well-serializable* if the following problems are in P:

- Given $n \in \mathbb{N}$ and $x \in \mathcal{C}_\mathcal{I}(n)$, find $y \in \mathcal{C}_\mathcal{I}(n)$ such that $x < y$ and there is no $y' \in \mathcal{C}_\mathcal{I}(n)$ with $x < y' < y$.
- Given $n \in \mathbb{N}$ and $x \in \mathcal{C}_\mathcal{I}(n)$, find $y \in \mathcal{C}_\mathcal{I}(n)$ such that $y < x$ and there is no $y' \in \mathcal{C}_\mathcal{I}(n)$ with $y < y' < x$.

In other words, finding the immediate successor and predecessor of a value $x \in \mathcal{C}_\mathcal{I}(n)$ is tractable for a well-serializable measure $\mathcal{I}$. Being able to calculate them enables us to adopt the following result for logic programs.

Lemma 4 (Lemma 3 in [48])**.** *Let* $\mathcal{I}$ *be a well-serializable inconsistency measure and* $X \subseteq \mathcal{P}$*. Let* $i \in \mathbb{N}$ *and* $\mathcal{C} \in \{\Sigma_i^p, \Pi_i^p\}$*. Then,*

- $\textsc{Upper}_\mathcal{I}^X$ *is* $\mathcal{C}$*-complete iff* $\textsc{Lower}_\mathcal{I}^X$ *is co-*$\mathcal{C}$*-complete.*

- *if* $\textsc{Upper}_{\mathcal{I}}^{X}$ *or* $\textsc{Lower}_{\mathcal{I}}^{X}$ *is in* $\mathcal{C}$*, then* $\textsc{Exact}_{\mathcal{I}}^{X}$ *is in* D_i^p.

It is straightforward to see that our proposed measures are indeed well-serializable, cf. Lemma 2.

Proposition 13. *For* $\mathcal{I} \in \{\mathcal{I}_{01}, \mathcal{I}_{\pm}, \mathcal{I}_{+}, \mathcal{I}_{-}, \mathcal{I}_{sd}, \mathcal{I}_{\#}\}$*,* $\mathcal{I}$ *is well-serializable.*

In the remainder of this section, we will give complexity results for the four problems for our measures on different classes of logic programs. Due to the insights above we will focus on the problem $\textsc{Upper}_{\mathcal{I}}^{X}$ which allows us to easily derive complexity bounds for the other classes.

6.1 The General Case

We first turn to the complexity of our measures on general disjunctive logic programs. Recall that $\mathcal{P}$ denotes the set of all disjunctive logic programs. As already mentioned, checking the consistency of a logic program in the general case is Σ_2^p-complete.

Theorem 2 ([18])**.** *Deciding whether a disjunctive logic program* $P \in \mathcal{P}$ *is consistent is* Σ_2^p*-complete.*

As inconsistency measures are supposed to generalize the concept of inconsistency by obeying the *Consistency* postulate, the above result already provides a lower bound for the complexity of $\textsc{Upper}_{\mathcal{I}}^{\mathcal{P}}$.

Proposition 14. *Let* $\mathcal{I}$ *be an inconsistency measure that satisfies* Consistency*. Then* $\textsc{Upper}_{\mathcal{I}}^{\mathcal{P}}$ *is* Σ_2^p*-hard.*

Proof. Due to *Consistency*, checking whether $\mathcal{I}(P) \leq 0$ corresponds to checking whether the program is consistent, which is Σ_2^p-hard as stated in Theorem 2. □

However, for every measure $\mathcal{I}$ we proposed in this paper, we obtain membership in Σ_2^p for $\textsc{Upper}_{\mathcal{I}}^{\mathcal{P}}$ via guess-and-check algorithms, yielding completeness for Σ_2^p.

Theorem 3. *For* $\mathcal{I} \in \{\mathcal{I}_{01}, \mathcal{I}_{\pm}, \mathcal{I}_{+}, \mathcal{I}_{-}, \mathcal{I}_{sd}, \mathcal{I}_{\#}\}$*,* $\textsc{Upper}_{\mathcal{I}}^{\mathcal{P}}$ *is* Σ_2^p*-complete.*

Proof. Hardness follows from Proposition 14. Completeness for $\textsc{Upper}_{\mathcal{I}_{01}}^{\mathcal{P}}$ is also obvious. Let (P, x) be an instance of $\textsc{Upper}_{\mathcal{I}}^{\mathcal{P}}$.

1. $\textsc{Upper}_{\mathcal{I}_{\pm}}^{\mathcal{P}} \in \Sigma_2^p$: We guess sets M, A of literals and a set $D \subseteq P$ of rules and verify in polynomial time that M is a model of $((P \cup A) \setminus D)^M$. Using an NP oracle we verify that there is no proper subset $M' \subseteq M$ that is also a model of $((P \cup A) \setminus D)^M$. Checking whether $|A| + |D| \leq x$ is also clearly in P. Hence, deciding $\textsc{Upper}_{\mathcal{I}_{\pm}}^{\mathcal{P}}$ is in Σ_2^p.

2. $\textsc{Upper}^{\mathcal{P}}_{\mathcal{I}_+} \in \Sigma^p_2$: analogous to above.

3. $\textsc{Upper}^{\mathcal{P}}_{\mathcal{I}_-} \in \Sigma^p_2$: analogous to above.

4. $\textsc{Upper}^{\mathcal{P}}_{\mathcal{I}_{sd}} \in \Sigma^p_2$: Similarly, guess a set M of literals and a set M' (a potential answer set of P^M). Checking that M' is a minimal model of P^M is in coNP, i. e., can be done using an NP oracle. Calculating $d_{sd}(M, M')$ is in P.

5. $\textsc{Upper}^{\mathcal{P}}_{\mathcal{I}_\#} \in \Sigma^p_2$: Guess a set M and verify that $M \in \mathrm{Ans}_{Con}(P^M)$. Counting the amount of complementary literals in M is in P.

□

Combining the above result with Lemma 3 and Lemma 4 we obtain the following picture on the computational complexity for general programs.

Corollary 3. *For $\mathcal{I} \in \{\mathcal{I}_{01}, \mathcal{I}_\pm, \mathcal{I}_+, \mathcal{I}_-, \mathcal{I}_{sd}, \mathcal{I}_\#\}$, $\textsc{Lower}^{\mathcal{P}}_{\mathcal{I}}$ is Π^p_2-complete, $\textsc{Exact}^{\mathcal{P}}_{\mathcal{I}}$ is in D^p_2 and $\textsc{Value}^{\mathcal{P}}_{\mathcal{I}}$ is in $FP^{\Sigma^p_2[\log n]}$.*

6.2 Disjunction-free Programs

We continue our investigation by considering only disjunction-free logic programs, i. e., programs with rules of the form

$$l_0 \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n.$$

Let $\mathcal{P}^{\not\vee} \subseteq \mathcal{P}$ be the set of all disjunction-free logic programs. For this class of programs we already have the following result from [18] pertaining to the complexity of deciding only consistency.

Theorem 4 ([18])**.** *Deciding whether an extended logic program P without disjunction is consistent is NP-complete.*

As before we can utilize this result to obtain a lower bound on the complexity of the problem $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}}$.

Proposition 15. *Let $\mathcal{I}$ be an inconsistency measure that satisfies* Consistency*. Then $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}}$ is NP-hard.*

We obtain the following theorem which is similar to our results above.

Theorem 5. *For $\mathcal{I} \in \{\mathcal{I}_{01}, \mathcal{I}_\pm, \mathcal{I}_+, \mathcal{I}_-, \mathcal{I}_{sd}, \mathcal{I}_\#\}$, $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}}$ is NP-complete.*

Proof. For all measures, hardness follows from Proposition 15. Completeness for $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}_{01}}$ is also obvious. Let (P, x) be an instance of $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}}$.

1. $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}_\pm} \in \mathsf{NP}$: We guess sets M, A of literals and a set $D \subseteq P$ of rules and verify in polynomial time that M is the unique model of $((P \cup A) \setminus D)^M$. Checking whether $|A| + |D| \leq x$ is also clearly in P. Hence, $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}_\pm}$ is in NP.

2. $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}_+} \in \mathsf{NP}$: analogous to above.

3. $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}_-} \in \mathsf{NP}$: analogous to above.

4. $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}_{sd}} \in \mathsf{NP}$: We can guess a set M of literals and determine the unique model M' of P^M in non-deterministic polynomial time. Computing $d_{sd}(M, M')$ can also be done in polynomial time.

5. $\textsc{Upper}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}_\#} \in \mathsf{NP}$: We can guess a set M of literals and check whether M is indeed the unique model of P^M in non-deterministic polynomial time. Determining the number of contradictory literals in M can also be done in polynomial time.

□

Corollary 4. *For $\mathcal{I} \in \{\mathcal{I}_{01}, \mathcal{I}_\pm, \mathcal{I}_+, \mathcal{I}_-, \mathcal{I}_{sd}, \mathcal{I}_\#\}$, $\textsc{Lower}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}}$ is coNP-complete, $\textsc{Exact}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}}$ is in $\mathsf{D}^{\mathsf{p}}_1$ and $\textsc{Value}^{\mathcal{P}^{\not\vee}}_{\mathcal{I}}$ is in $\mathsf{FP}^{\mathsf{NP}[\log n]}$.*

6.3 Stratified Programs

We now consider *stratified programs* as another special case, see e. g. [2].

Definition 23. Let P be a disjunction-free program consisting of rules of the form

$$l_0 \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n. \tag{7}$$

We call P *stratified* if there is a mapping $\|.\| : \mathcal{L}(P) \to \mathbb{N}$ such that for each rule $r \in P$ of the form (7) the following holds:

1. $\|l_i\| \leq \|l_0\|$ for each $1 \leq i \leq m$,
2. $\|l_j\| < \|l_0\|$ for each $m + 1 \leq j \leq n$.

Let $\mathcal{P}^{\|\cdot\|}$ be the set of all stratified programs. A stratified program has a unique answer set which can be computed in polynomial time [2, 14] which gives them many advantages in practical applications. A simple corollary of this is the following observation.

Corollary 5 (implied by [2])**.** *Deciding whether a stratified logic program P is consistent is in P.*

The results so far and the above observation may lead to the conjecture that the decision problem $\textsc{Upper}_{\mathcal{I}}^{\mathcal{P}^{\|\cdot\|}}$ for our measures can also be solved in polynomial time. At least for the measures $\mathcal{I}_{01}$ and $\mathcal{I}_{\#}$, this conjecture is true.

Theorem 6. $\textsc{Upper}_{\mathcal{I}_{01}}^{\mathcal{P}^{\|\cdot\|}} \in \mathsf{P}$ *and* $\textsc{Upper}_{\mathcal{I}_{\#}}^{\mathcal{P}^{\|\cdot\|}} \in \mathsf{P}$.

Proof. We can calculate the unique answer set M of P and count the number of complementary literals in M in polynomial time. □

Corollary 6. $\textsc{Lower}_{\mathcal{I}_{01}}^{\mathcal{P}^{\|\cdot\|}} \in \mathsf{P}$, $\textsc{Exact}_{\mathcal{I}_{01}}^{\mathcal{P}^{\|\cdot\|}} \in \mathsf{P}$, $\textsc{Value}_{\mathcal{I}_{01}}^{\mathcal{P}^{\|\cdot\|}} \in \mathsf{FP}$, $\textsc{Lower}_{\mathcal{I}_{\#}}^{\mathcal{P}^{\|\cdot\|}} \in \mathsf{P}$, $\textsc{Exact}_{\mathcal{I}_{\#}}^{\mathcal{P}^{\|\cdot\|}} \in \mathsf{P}$, $\textsc{Value}_{\mathcal{I}_{\#}}^{\mathcal{P}^{\|\cdot\|}} \in \mathsf{FP}$

Unfortunately, the decision problem $\textsc{Upper}_{\mathcal{I}}^{\mathcal{P}^{\|\cdot\|}}$ for the other measures remains NP-complete.

Theorem 7. $\textsc{Upper}_{\mathcal{I}_{\pm}}^{\mathcal{P}^{\|\cdot\|}}$ *and* $\textsc{Upper}_{\mathcal{I}_{-}}^{\mathcal{P}^{\|\cdot\|}}$ *are* NP*-complete.*

Proof. Membership in NP is given through Theorem 5 and the fact that every stratified program is also a disjunction-free program. Hardness follows from Theorems 10 and 11 (see the next section) and the fact that every classical disjunction free program is also a stratified program. □

Theorem 8. $\textsc{Upper}_{\mathcal{I}_{+}}^{\mathcal{P}^{\|\cdot\|}}$ *and* $\textsc{Upper}_{\mathcal{I}_{sd}}^{\mathcal{P}^{\|\cdot\|}}$ *are* NP*-complete.*

Proof. We show NP-hardness by reduction of the Set Cover Problem, which is known to be NP-complete [31]. Given a set of integers $\mathcal{U} = \{1, ..., m\}$, a set of subsets $\mathcal{S} = \{S_1, ..., S_t\}$, $S_j \subseteq \mathcal{U}$ for each j, and an integer k, the Set Cover Problem asks whether there are at most k sets in $\mathcal{S}$ that cover $\mathcal{U}$.

Given an instance $\mathcal{U} = \{1, ..., m\}$, $\mathcal{S} = \{S_1, ..., S_t\}$, $S_j \subseteq \mathcal{U}$ for each j of the Set Cover Problem and an integer k, we construct the following program P:

- For each number $i \in \mathcal{U}$, we construct two rules r_i and $\neg r_i$ having literals p_i and $\neg p_i$ as head, respectively.
- For each set $S_j \in \mathcal{S}$, we let an atom a_j appear in $neg(r_i)$ and $neg(\neg r_i)$ if and only $i \in S_j$.

Example 15. For $\mathcal{U} = \{1, 2, 3\}$ and $\mathcal{S} = \{S_1, S_2, S_3\} = \{\{1, 2\}, \{2, 3\}, \{1, 3\}\}$, we obtain the following program:

$$
\begin{aligned}
p_1 &\leftarrow \text{not } a_1, \text{not } a_3.\\
\neg p_1 &\leftarrow \text{not } a_1, \text{not } a_3.\\
p_2 &\leftarrow \text{not } a_1, \text{not } a_2.\\
\neg p_2 &\leftarrow \text{not } a_1, \text{not } a_2.\\
p_3 &\leftarrow \text{not } a_2, \text{not } a_3.\\
\neg p_3 &\leftarrow \text{not } a_2, \text{not } a_3.
\end{aligned}
$$

Clearly, this program is stratified. Furthermore, if A is a set of facts such that $P \cup A$ is consistent, then we see as above that A corresponds to a cover of $\mathcal{U}$. We obtain hardness for $\mathcal{I}_+$.

Similarly, if P^M is consistent, then M corresponds to a cover of $\mathcal{U}$. Furthermore P^M will always have $\emptyset$ as the minimal model. So, such a set M with $d_{sd}(M, \mathrm{Ans}_{Con}(P^M)) \leq k$ corresponds to a cover with at most k sets. Hence, we obtain hardness for $\textsc{Upper}^{\mathcal{P}^{\|\cdot\|}}_{\mathcal{I}_{sd}}$. □

Corollary 7. *For $\mathcal{I} \in \{\mathcal{I}_\pm, \mathcal{I}_+, \mathcal{I}_-, \mathcal{I}_{sd}\}$, $\textsc{Lower}^{\mathcal{P}^{\|\cdot\|}}_{\mathcal{I}}$ is coNP-complete, $\textsc{Exact}^{\mathcal{P}^{\|\cdot\|}}_{\mathcal{I}}$ is in D^p_1 and $\textsc{Value}^{\mathcal{P}^{\|\cdot\|}}_{\mathcal{I}}$ is in $\mathsf{FP}^{\mathsf{NP}[\log n]}$.*

6.4 Classical Disjunction-free Programs

Finally, we investigate the computational complexity of our measures on the easiest class of logic programs, namely classical disjunction-free programs P consisting of rules of the form

$$l_0 \leftarrow l_1, \ldots, l_m. \tag{8}$$

with literals $l_0, \ldots, l_m$. Let $\mathcal{P}^{\not\exists\not\forall} \subseteq \mathcal{P}$ be the set of all classical disjunction-free programs.

Recall that every classical disjunction-free program has a unique answer set that can be computed in polynomial time. This also implies the following observation.

Corollary 8. *Deciding whether a classical disjunction-free program is consistent is in P.*

As every classical disjunction-free program is also a stratified program, i. e. $\mathcal{P}^{\not\exists\not\forall} \subseteq \mathcal{P}^{\|\cdot\|}$, Theorem 6 and Corollary 6 already give us the following results.

Corollary 9. $\textsc{Upper}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}_{01}} \in \mathsf{P}$, $\textsc{Upper}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}_\#} \in \mathsf{P}$, $\textsc{Lower}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}_{01}} \in \mathsf{P}$, $\textsc{Exact}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}_{01}} \in \mathsf{P}$, $\textsc{Value}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}_{01}} \in \mathsf{FP}$, $\textsc{Lower}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}_\#} \in \mathsf{P}$, $\textsc{Exact}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}_\#} \in \mathsf{P}$, $\textsc{Value}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}_\#} \in \mathsf{FP}$.

Also the problems related to the measures $\mathcal{I}_+$ and $\mathcal{I}_{sd}$ turn out to be feasible for classical disjunction-free programs.

Theorem 9. *For $\mathcal{I} \in \{\mathcal{I}_+, \mathcal{I}_{sd}\}$, $\textsc{Upper}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}} \in \mathsf{P}$, $\textsc{Lower}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}} \in \mathsf{P}$, $\textsc{Exact}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}} \in \mathsf{P}$, $\textsc{Value}^{\mathcal{P}^{\not\exists\not\forall}}_{\mathcal{I}} \in \mathsf{FP}$.*

Proof. Note that if a classical program P is inconsistent there is no A such that $P \cup A$ becomes consistent. Therefore, for every $P \in \mathcal{P}^{\not\exists\not\forall}$ we have either $\mathcal{I}_+(P) = 0$ or $\mathcal{I}_+(P) = \infty$. We can decide $\mathcal{I}_+(P) \leq 0$ in polynomial time due to Corollary 8. Furthermore, note that for a classical program P we have $P = P^M$ for every set of literals M. It follows that for every $P \in \mathcal{P}^{\not\exists\not\forall}$ we have either $\mathcal{I}_{sd}(P) = 0$ or $\mathcal{I}_{sd}(P) = \infty$ as well.

The remaining cases follow from these observations. □

Unfortunately, for measures $\mathcal{I}_\pm$ and $\mathcal{I}_-$, all problems remain infeasible even for the class $\mathcal{P}^{\not\text{not}\not/}$.

Theorem 10. $\textsc{Upper}^{\mathcal{P}^{\not\text{not}\not/}}_{\mathcal{I}_\pm}$ *is* **NP***-complete.*

Proof. Membership is clear since this is a special case of Theorem 5. Again, we show NP-hardness using the Set Cover Problem [31]. Thus, we let $\mathcal{U} = \{1, ..., m\}$ be a set, $\mathcal{S} = \{S_1, ..., S_t\}$ be a set of subsets of $\mathcal{U}$ and k an integer. Recall that the Set Cover Problem asks whether there are at most k sets in $\mathcal{S}$ that cover $\mathcal{U}$.

Suppose we were given an integer k and could solve $\textsc{Upper}^{\mathcal{P}^{\not\text{not}\not/}}_{\mathcal{I}_\pm}$. We construct the following classical disjunction-free program P:

- For each number $i \in \mathcal{U}$, we construct two rules r_i and $\neg r_i$ having literals p_i and $\neg p_i$ as head, respectively.
- For each set $S_j \in \mathcal{S}$, we consider an atom a_j, construct a fact "a_j." and let a_j appear in $pos(r_i)$ and $pos(\neg r_i)$ if and only $i \in S_j$.

Example 16. For $\mathcal{U} = \{1, 2, 3\}$ and $\mathcal{S} = \{S_1, S_2, S_3\} = \{\{1, 2\}, \{2, 3\}, \{1, 3\}\}$, we obtain the following classical disjunction-free program:

$$
\begin{aligned}
a_1.\quad a_2.\quad a_3.&\\
p_1 \leftarrow & a_1,\ a_3.\\
\neg p_1 \leftarrow & a_1,\ a_3.\\
p_2 \leftarrow & a_1,\ a_2.\\
\neg p_2 \leftarrow & a_1,\ a_2.\\
p_3 \leftarrow & a_2,\ a_3.\\
\neg p_3 \leftarrow & a_2,\ a_3.
\end{aligned}
$$

If D is a subset of the facts $\{a_1., \ldots, a_t.\}$ such that $P \setminus D$ is consistent, then each pair "r_i" and "$\neg r_i$" of rules is not applicable anymore. Thus, D corresponds to a cover of $\mathcal{U}$. It is only left to show that we can assume the set D of deleted rules to not contain any of the "r_i" and "$\neg r_i$". However, this is clear since deleting one of them resolves at most one conflict and can be mimicked by removing the corresponding fact from an atom which appears in the rules. (This also corresponds to adding one set S_j that covers at least one additional integer in $\mathcal{U}$ which is trivial at any point.) Moreover, the set A of added rules can be assumed to be empty since we constructed a classical program. □

We obtain the same result with the same proof for $\mathcal{I}_-$.

Theorem 11. $\textsc{Upper}^{\mathcal{P}^{\not\text{not}\not/}}_{\mathcal{I}_-}$ *is* **NP***-complete.*

Corollary 10. *For $\mathcal{I} \in \{\mathcal{I}_{\pm}, \mathcal{I}_{-}\}$,* $\textsc{Lower}_{\mathcal{I}}^{\mathcal{P}^{\not\emptyset}}$ *is coNP-complete,* $\textsc{Exact}_{\mathcal{I}}^{\mathcal{P}^{\not\emptyset}}$ *is in* D_1^p *and* $\textsc{Value}_{\mathcal{I}}^{\mathcal{P}^{\not\emptyset}}$ *is in* $\mathsf{FP}^{\mathsf{NP}[\log n]}$.

Table 2 summarizes the results of this section.

		$\mathcal{Q} = \mathcal{P}$	$\mathcal{Q} = \mathcal{P}^{\not\emptyset}$	$\mathcal{Q} = \mathcal{P}^{\|\cdot\|}$	$\mathcal{Q} = \mathcal{P}^{\not\emptyset}$
$\mathcal{I}_{\pm}$	$\textsc{Upper}_{\mathcal{I}_{\pm}}^{\mathcal{Q}}$	Σ_2^p-c	NP-c	NP-c	NP-c
	$\textsc{Lower}_{\mathcal{I}_{\pm}}^{\mathcal{Q}}$	Π_2^p-c	coNP-c	coNP-c	coNP-c
	$\textsc{Exact}_{\mathcal{I}_{\pm}}^{\mathcal{Q}}$	D_2^p	D_1^p	D_1^p	D_1^p
	$\textsc{Value}_{\mathcal{I}_{\pm}}^{\mathcal{Q}}$	$\mathsf{FP}^{\Sigma_2^p[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$
$\mathcal{I}_{-}$	$\textsc{Upper}_{\mathcal{I}_{-}}^{\mathcal{Q}}$	Σ_2^p-c	NP-c	NP-c	NP-c
	$\textsc{Lower}_{\mathcal{I}_{-}}^{\mathcal{Q}}$	Π_2^p-c	coNP-c	coNP-c	coNP-c
	$\textsc{Exact}_{\mathcal{I}_{-}}^{\mathcal{Q}}$	D_2^p	D_1^p	D_1^p	D_1^p
	$\textsc{Value}_{\mathcal{I}_{-}}^{\mathcal{Q}}$	$\mathsf{FP}^{\Sigma_2^p[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$
$\mathcal{I}_{+}$	$\textsc{Upper}_{\mathcal{I}_{+}}^{\mathcal{Q}}$	Σ_2^p-c	NP-c	NP-c	P
	$\textsc{Lower}_{\mathcal{I}_{+}}^{\mathcal{Q}}$	Π_2^p-c	coNP-c	coNP-c	P
	$\textsc{Exact}_{\mathcal{I}_{+}}^{\mathcal{Q}}$	D_2^p	D_1^p	D_1^p	P
	$\textsc{Value}_{\mathcal{I}_{+}}^{\mathcal{Q}}$	$\mathsf{FP}^{\Sigma_2^p[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$	FP
$\mathcal{I}_{sd}$	$\textsc{Upper}_{\mathcal{I}_{sd}}^{\mathcal{Q}}$	Σ_2^p-c	NP-c	NP-c	P
	$\textsc{Lower}_{\mathcal{I}_{sd}}^{\mathcal{Q}}$	Π_2^p-c	coNP-c	coNP-c	P
	$\textsc{Exact}_{\mathcal{I}_{sd}}^{\mathcal{Q}}$	D_2^p	D_1^p	D_1^p	P
	$\textsc{Value}_{\mathcal{I}_{sd}}^{\mathcal{Q}}$	$\mathsf{FP}^{\Sigma_2^p[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$	FP
$\mathcal{I}_{\#}$	$\textsc{Upper}_{\mathcal{I}_{\#}}^{\mathcal{Q}}$	Σ_2^p-c	NP-c	P	P
	$\textsc{Lower}_{\mathcal{I}_{\#}}^{\mathcal{Q}}$	Π_2^p-c	coNP-c	P	P
	$\textsc{Exact}_{\mathcal{I}_{\#}}^{\mathcal{Q}}$	D_2^p	D_1^p	P	P
	$\textsc{Value}_{\mathcal{I}_{\#}}^{\mathcal{Q}}$	$\mathsf{FP}^{\Sigma_2^p[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$	FP	FP
$\mathcal{I}_{01}$	$\textsc{Upper}_{\mathcal{I}_{01}}^{\mathcal{Q}}$	Σ_2^p-c	NP-c	P	P
	$\textsc{Lower}_{\mathcal{I}_{01}}^{\mathcal{Q}}$	Π_2^p-c	coNP-c	P	P
	$\textsc{Exact}_{\mathcal{I}_{01}}^{\mathcal{Q}}$	D_2^p	D_1^p	P	P
	$\textsc{Value}_{\mathcal{I}_{01}}^{\mathcal{Q}}$	$\mathsf{FP}^{\Sigma_2^p[\log n]}$	$\mathsf{FP}^{\mathsf{NP}[\log n]}$	FP	FP

Table 2: Computational complexity of various problems related to our inconsistency measures

7 Summary and Discussion

In this paper, we addressed the challenge of measuring inconsistency in ASP by critically reviewing the propositional framework of inconsistency measurement and taking non-monotonicity into account. We developed novel rationality postulates and measures that are more apt for analyzing inconsistency in ASP than existing approaches. Intuitively, some of our measures take the effort

needed to restore the consistency of programs into account ($\mathcal{I}_{\pm}$, $\mathcal{I}_{+}$, $\mathcal{I}_{-}$), and our results show that it does not matter whether this is done on the level of the original program or on the level of the reduct. Others measure inconsistency in terms of the quality of the produced output, e. g. $\mathcal{I}_{\#}$ which considers the minimal number of inconsistencies in an answer set. We showed that our new measures comply with many of our rationality postulates and illustrated their usage.

To the best of our knowledge, measuring inconsistency in extended logic programs under the answer set semantics has not been addressed before. The closest related works are by Madrid and Ojeda-Aciego, see e. g. [37, 38], who address inconsistencies in residuated logic programs under fuzzy answer set semantics. In their setting, rules such as (1) are augmented with fuzzy values in $[0, 1]$ (or some arbitrary lattice) and inconsistency is measured by considering minimal changes in the values to restore the existence of fuzzy stable models. However, Madrid and Ojeda-Aciego do not discuss the propositional case and rationality postulates.

Inconsistency measures have numerous applications, for instance in belief revision where the degree of inconsistency of new information may provide useful guidance as to whether the new information should be accepted or not, or in belief merging where the degree of inconsistency may be used to decide whether the views of a particular agent should be taken into account. The overview paper [28] contains an analysis of these and various other potential applications. The results of this paper pave the way for similar applications in the context of ASP, which has become a popular language for declarative problem solving.

In future work, we would like to extend our analysis to more general classes of logic programs, e. g., programs with choice rules, weight constraints and aggregates. For an overview on these extensions see [20]. It would also be interesting to see whether our measures, or similar ones, can be applied to other non-monotonic formalisms, like default logic [41] or autoepistemic logic [39]. In another recent paper [11], we explored the issues of minimal inconsistent sets in general non-monotonic formalisms in more depth. There, we also hinted to possible applications for inconsistency measurement which is also part of ongoing work.

Acknowledgements

We would like to thank the reviewers of the present paper and of [50] for useful comments which also helped to improve the current paper. This work has been partially funded by the DFG Research Training Group 1763 and DFG project BR 1817/7-2.

References

[1] Meriem Ammoura, Yakoub Salhi, Brahim Oukacha, and Badran Raddaoui. On an mcs-based inconsistency measure. *International Journal of Approximate Reasoning*, 2016.

[2] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.

[3] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *The Knowledge Engineering Review*, 26(4):365–410, 2011.

[4] Salem Benferhat, Didier Dubois, and Henri Prade. A local approach to reasoning under inconsisteny in stratified knowledge bases. In *Proceedings of the Third European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU'95)*, pages 36–43, 1995.

[5] Philippe Besnard. Revisiting postulates for inconsistency measures. In *Proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA'14)*, pages 383–396, 2014.

[6] Philippe Besnard. Forgetting-based inconsistency measure. In *Proceedings of the 10th International Conference on Scalable Uncertainty Management (SUM'16)*, pages 331–337, 2016.

[7] Jean-Yves Béziau, Walter Carnielli, and Dov Gabbay, editors. *Handbook of Paraconsistency*. College Publications, London, 2007.

[8] Howard A. Blair and V.S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68(2):135–154, 1989.

[9] Ronald J. Brachman and Hector J. Levesque. *Knowledge Representation and Reasoning*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann Publishers, 2004.

[10] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[11] Gerhard Brewka, Matthias Thimm, and Markus Ulbricht. Strong inconsistency in nonmonotonic reasoning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, August 2017.

[12] L. Cholvy and A. Hunter. Information fusion in logic: A brief overview. In *Qualitative and Quantitative Practical Reasoning (ECSQARU'97/FAPR'97)*, volume 1244 of *Lecture Notes in Computer Science*, pages 86–95. Springer, 1997.

[13] Stefania Costantini. On the existence of stable models of non-stratified logic programs. *Theory and Practice of Logic Programming*, 6(1-2):169–212, January 2006.

[14] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[15] James P. Delgrande, Torsten Schaub, Hans Tompits, and Stefan Woltran. Belief revision of logic programs under answer set semantics. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning*, pages 411–421, 2008.

[16] Didier Dubois, Jérôme Lang, and Henri Prade. Inconsistency in possibilistic knowledge bases: To live with it or not live with it. In L.A. Zadeh and J. Kacprzyk, editors, *Fuzzy Logic for the Management of Uncertainty* , pages 335–351. Wiley, New York, 1992.

[17] Didier Dubois and Henri Prade. A possibilistic analysis of inconsistency. In *Scalable Uncertainty Management - 9th International Conference, SUM 2015, Québec City, QC, Canada, September 16-18, 2015. Proceedings*, pages 347–353, 2015.

[18] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[19] Dov M. Gabbay, Laura Giordano, Alberto Martelli, and Nicola Olivetti. Hypothetical updates, priority and inconsistency in a logic programming language. In *Logic Programming and Nonmonotonic Reasoning, Third International Conference, LPNMR'95, Lexington, KY, USA, June 26-28, 1995, Proceedings*, pages 203–216, 1995.

[20] Martin Gebser and Torsten Schaub. Modeling and language extensions. *AI Magazine*, 37(3):33–44, 2016.

[21] M. Gelfond and N. Leone. Logic programming and knowledge representation – the A-Prolog perspective. *Artificial Intelligence*, 138(1–2):3–38, 2002.

[22] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[23] John Grant. Classifications for inconsistent theories. *Notre Dame Journal of Formal Logic*, 19(3):435–444, 1978.

[24] John Grant and Anthony Hunter. Measuring Inconsistency in Knowledgebases. *Journal of Intelligent Information Systems*, 27:159–184, 2006.

[25] John Grant and Anthony Hunter. Measuring the good and the bad in inconsistent information. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2632–2637, 2011.

[26] John Grant and Anthony Hunter. Analysing inconsistent information using distance-based measures. *International Journal of Approximate Reasoning*, In press, 2016.

[27] Sven Ove Hansson. *A Textbook of Belief Dynamics*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[28] A. Hunter and S. Konieczny. Approaches to Measuring Inconsistent Information. In *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*, pages 189–234. Springer International Publishing, 2004.

[29] A. Hunter and S. Konieczny. Measuring inconsistency through minimal inconsistent sets. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'2008)*, pages 358–366. AAAI Press, 2008.

[30] Said Jabbour, Yue Ma, Badran Raddaoui, Lakhdar Sais, and Yakoub Salhi. A MIS partition based framework for measuring inconsistency. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR'16)*, pages 84–93, 2016.

[31] Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103, 1972.

[32] Sébastien Konieczny, Jérôme Lang, and Pierre Marquis. Reasoning under inconsistency: the forgotten connective. In *Proceedings of IJCAI-2005*, pages 484–489, 2005.

[33] Sebastien Konieczny and Ramon Pino Perez. Logic based merging. *Journal of Philosophical Logic*, 40:239–270, 2011.

[34] Jerome Lang and Pierre Marquis. Reasoning under inconsistency: A forgetting-based approach. *Artificial Intelligence*, 174(12–13):799–823, 2010.

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

[36] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994*, pages 23–37, 1994.

[37] Nicolás Madrid and Manuel Ojeda-Aciego. Measuring instability in normal residuated logic programs: Adding information. In *FUZZ-IEEE 2010, IEEE International Conference on Fuzzy Systems, Barcelona, Spain, 18-23 July, 2010, Proceedings*, pages 1–7, 2010.

[38] Nicolas Madrid and Manuel Ojeda-Aciego. Measuring inconsistency in fuzzy answer set semantics. *IEEE Transactions on Fuzzy Systems*, 19(4):605–622, 2011.

[39] Robert C. Moore. Autoepistemic logic revisited. *Artif. Intell.*, 59(1-2):27–30, 1993.

[40] Kedian Mu. Responsibility for inconsistency. *International Journal of Approximate Reasoning*, 61:43–60, 2015.

[41] Raymond Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

[42] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[43] Claudia Schulz, Ken Satoh, and Francesca Toni. Characterising and explaining inconsistency in logic programs. In Francesco Calimeri, Giovambattista Ianni, and Miroslaw Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning: 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings*, pages 467–479. Springer International Publishing, Cham, 2015.

[44] Matthias Thimm. Inconsistency Measures for Probabilistic Logics. *Artificial Intelligence*, 197:1–24, 2013.

[45] Matthias Thimm. On the compliance of rationality postulates for inconsistency measures: A more or less complete picture. *Künstliche Intelligenz*, 2016.

[46] Matthias Thimm. On the expressivity of inconsistency measures. *Artificial Intelligence*, 234:120–151, 2016.

[47] Matthias Thimm. Stream-based inconsistency measurement. *International Journal of Approximate Reasoning*, 68:68–87, 2016.

[48] Matthias Thimm and Johannes Peter Wallner. Some complexity results on inconsistency measurement. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR'16)*, pages 114–124, 2016.

[49] M. Truszczynski. Strong and uniform equivalence of nonmonotonic theories - an algebraic approach. *Annals of Mathematics and Artificial Intelligence*, 48(3–4):245–265, 2006.

[50] Markus Ulbricht, Matthias Thimm, and Gerhard Brewka. Measuring inconsistency in answer set programs. In *Logics in Artificial Intelligence - 15th European Conference, JELIA 2016, Larnaca, Cyprus, November 9-11, 2016, Proceedings*, pages 577–583, 2016.