An Algorithm for the Contension Inconsistency Measure Using Reductions to Answer Set Programming

Isabelle Kuhlmann(✉) and Matthias Thimm

University of Koblenz-Landau, Koblenz, Germany
{iskuhlmann,thimm}@uni-koblenz.de

Abstract. We present an algorithm for determining inconsistency degrees wrt. the contension inconsistency measure [7] which utilizes three-valued logic to determine the minimal number of atoms that are assigned truth value B (paradoxical/both true and false). Our algorithm is based on an answer set programming encoding for checking for upper bounds and a binary search algorithm on top of that. We experimentally show that the new algorithm significantly outperforms the state of the art.

Keywords: Inconsistency Measurement · Answer set programming · Contension inconsistency measure

1 Introduction

Dealing with inconsistent information is an important aspect in rational accounts to formal reasoning. In applications such as decision-support systems, a knowledge base is usually compiled by merging the formalised knowledge of many different experts. It is unavoidable that different experts contradict each other and that the merged knowledge base becomes inconsistent. One way of dealing with inconsistent information is to abandon classical inference and define new ways of reasoning. Some examples of such formalisms are, e. g., paraconsistent logics [2], default logic [12], answer set programming [3], and computational models of argumentation [1]. Moreover, the field of belief revision [9] deals with the particular case of inconsistencies in dynamic settings.

The field of *Inconsistency Measurement*—see the seminal work [6] and the recent book [8]—provides an *analytical* perspective on the issue of inconsistency. An *inconsistency measure* is a function that maps a knowledge base to a non-negative real number, the interpretation of that number being that larger values indicate a larger inconsistency within the knowledge base. The field of inconsistency measurement has proposed a series of different approaches to measure inconsistency, focusing on aspects such as minimal inconsistent subsets [4,10], or non-classical semantics [7,13], see [14] for an overview.

In this paper, we are considering algorithmic problems involving the contension inconsistency measure from [7]. This measure uses Priest's three-valued

J. Davis and K. Tabia (Eds.): SUM 2020, LNAI 12322, pp. 289–296, 2020.
https://doi.org/10.1007/978-3-030-58449-8_23

logic [11] to determine the minimal number of atoms in the language that are "conflicting" in the knowledge base under consideration (we will provide formal details in Sect. 2). In [15] it has been shown that the problem of deciding whether a certain value is an upper bound for the inconsistency degree wrt. the contension measure in NP-complete. Although this is an intractable problem, it still belongs to the rather "easier" problems in inconsistency measurement, as the corresponding decision problems wrt. many other measures are even higher in the polynomial hierarchy [15]. In this paper, we are presenting an algorithm for determining the inconsistency degree wrt. the contension measure of an arbitrary knowledge base using answer set programming (ASP) [3]. The latter is a declarative problem solving formalism that is suitable for addressing NP-hard problems (and beyond). More specifically, the contributions of this work are the following:

1. We present an ASP encoding of the problem whether a certain number is an upper bound for the inconsistency degree and integrate it into a binary search algorithm for determining the actual value (Sect. 3).
2. We report on some preliminary experiments that show that this algorithm significantly outperforms the state of the art (Sect. 4).

In addition, Sect. 2 covers the necessary preliminaries and Sect. 5 concludes the work by giving an overview of the contributions and possible future work.

2 Preliminaries

Let At be some fixed set of propositions and let $\mathcal{L}(\mathsf{At})$ be the corresponding propositional language constructed using the usual connectives $\wedge$ (*conjunction*), $\vee$ (*disjunction*), and $\neg$ (*negation*).

Definition 1. *A knowledge base $\mathcal{K}$ is a finite set of formulas $\mathcal{K} \subseteq \mathcal{L}(\mathsf{At})$. Let $\mathbb{K}$ be the set of all knowledge bases.*

If X is a formula or a set of formulas we write $\mathsf{At}(X)$ to denote the set of propositions appearing in X.

Semantics for a propositional language is given by *interpretations* where an *interpretation* ω on At is a function $\omega : \mathsf{At} \to \{\mathsf{true}, \mathsf{false}\}$. Let $\Omega(\mathsf{At})$ denote the set of all interpretations for At. An interpretation ω *satisfies* (or is a *model* of) a proposition $a \in \mathsf{At}$, denoted by $\omega \models a$, if and only if $\omega(a) = \mathsf{true}$. The satisfaction relation $\models$ is extended to formulas in the usual way.

For $\Phi \subseteq \mathcal{L}(\mathsf{At})$ we also define $\omega \models \Phi$ if and only if $\omega \models \phi$ for every $\phi \in \Phi$. A formula or set of formulas X_1 *entails* another formula or set of formulas X_2, denoted by $X_1 \models X_2$, if and only if $\omega \models X_1$ implies $\omega \models X_2$. If there is no ω with $\omega \models X$ we also write $X \models \bot$ and say that X is *inconsistent*.

2.1 The Contension Inconsistency Measure

Let $\mathbb{R}^{\infty}_{\geq 0}$ be the set of non-negative real values including infinity. The most general form of an inconsistency measure is as follows.

Definition 2. *An* inconsistency measure $\mathcal{I}$ *is a function* $\mathcal{I} : \mathbb{K} \to \mathbb{R}^{\infty}_{\geq 0}$ *that satisfies* $\mathcal{I}(\mathcal{K}) = 0$ *if and only if* $\mathcal{K}$ *is consistent, for all* $\mathcal{K} \in \mathbb{K}$.

The intuition we intend to be behind any concrete approach to inconsistency measure $\mathcal{I}$ is that a larger value $\mathcal{I}(\mathcal{K})$ for a knowledge base $\mathcal{K}$ indicates more severe inconsistency in $\mathcal{K}$ than lower values. Moreover, we reserve the minimal value (0) to indicate the complete absence of inconsistency.

With regard to an inconsistency measure $\mathcal{I}$, $\textsc{Exact}_{\mathcal{I}}$ denotes the problem of deciding whether a given value $a \in \mathbb{R}^{\infty}_{\geq 0}$ is the inconsistency value $\mathcal{I}(\mathcal{K})$ of a knowledge base $\mathcal{K}$ [15]. $\textsc{Upper}_{\mathcal{I}}$ and $\textsc{Lower}_{\mathcal{I}}$ denote the problems whether a given value $a \in \mathbb{R}^{\infty}_{\geq 0}$ is an upper or a lower bound of $\mathcal{I}(\mathcal{K})$, respectively. For any function $\mathcal{I}$ according to Definition 2 that satisfies $\mathcal{I}(\mathcal{K}) = 0$ if and only if $\mathcal{K}$ is consistent, the decision problems $\textsc{Upper}_{\mathcal{I}}$ and $\textsc{Exact}_{\mathcal{I}}$ are NP-hard. $\textsc{Lower}_{\mathcal{I}}$ is coNP-hard [15]. Moreover, $\textsc{Value}_{\mathcal{I}}$ is the natural function problem which returns the value of $\mathcal{I}(\mathcal{K})$ for a given knowledge base $\mathcal{K}$.

In [7], Grant and Hunter introduce an inconsistency measure based on semantics of Priest's three-valued logic [11]. In addition to *true* (T) and *false* (F), this logic includes a third value which indicates *paradoxical*, or *both true and false* (B). Table 1 shows the truth tables for this logic.

Table 1. Truth tables for Priest's propositional three-valued logic [11]

x	y	$x \wedge y$	$x \vee y$
T	T	T	T
T	B	B	T
T	F	F	T
B	T	B	T
B	B	B	B
B	F	F	B
F	T	F	T
F	B	F	B
F	F	F	F

x	$\neg x$
T	F
B	B
F	T

Let i be a three-valued interpretation, i. e., a function that assigns one of the three truth values to each atom in a knowledge base $\mathcal{K}$, denoted as $i : \mathsf{At}(\mathcal{K}) \mapsto \{T, F, B\}$. The domain of a certain interpretation i can be divided into two groups corresponding to their truth value [7]. More specifically, there is the group of atoms which are assigned a classical truth value (T or F), and there is the group of atoms which are assigned B. The former is defined as follows:

$$\mathsf{Binarybase}(i) = \{\alpha \mid i(\alpha) = T \text{ or } i(\alpha) = F\}$$

Because the other group comprises those atoms which take part in conflicts, it is denoted

$$\mathsf{Conflictbase}(i) = \{\alpha \mid i(\alpha) = B\}.$$

Further, a *model* is defined as an interpretation where each formula ϕ in $\mathcal{K}$ is assigned either T or B. Thus, the set of models is defined as follows:

$$\mathsf{Models}(\mathcal{K}) = \{i \mid \forall \phi \in \mathcal{K}, i(\phi) = T \text{ or } i(\phi) = B\}$$

Example 1. Consider knowledge base $\mathcal{K}_1 = \{a \wedge b, \neg a \vee b, \neg b \wedge \neg c\}$. A model of $\mathcal{K}_1$ is the interpretation which assigns T to a, F to c, and B to b, denoted i_1. Consequently, Binarybase and Conflictbase wrt. i_1 are the following:

$$\mathsf{Binarybase}(i_1) = \{a, c\} \qquad \mathsf{Conflictbase}(i_1) = \{b\}$$

There exist also other models, for example the interpretation that assigns B to all $x \in \{a, b, c\}$ is a model of every knowledge base.

The previous definitions allows us to formulate the contension inconsistency measure $\mathcal{I}_c$ wrt. a knowledge base $\mathcal{K}$ as

$$\mathcal{I}_c(\mathcal{K}) = \min\{|\mathsf{Conflictbase}(i)| \mid i \in \mathsf{Models}(\mathcal{K})\}.$$

Consequently, $\mathcal{I}_c$ describes the minimum number of atoms in $\mathcal{K}$ that are assigned truth value B. Considering the exemplary knowledge base $\mathcal{K}_1$ presented in Example 1, we can easily see that it is inconsistent. More specifically, the first and third formula, $a \wedge b$ and $\neg b \wedge \neg c$, respectively, contradict each other in the sense of classical propositional logic. Since at least b (i. e., one atom) must be assigned the truth value B to make the knowledge base consistent in three-valued logic, the minimal size of Conflictbase is 1. Thus, $\mathcal{I}_c(\mathcal{K}_1) = 1$.

It has been shown [15] that $\textsc{Upper}_{\mathcal{I}_c}$ is NP-complete, $\textsc{Lower}_{\mathcal{I}_c}$ is coNP-complete, $\textsc{Exact}_{\mathcal{I}_c}$ is DP-complete, and $\textsc{Value}_{\mathcal{I}_c}$ is $\mathsf{FP}^{\mathsf{NP}[\log]}$-complete. To the best of our knowledge, the currently only existing algorithm for computing $\mathcal{I}_c(\mathcal{K})$ is the one given in TweetyProject[1]. This algorithm follows a naive approach by searching for a solution in a brute force fashion. The given knowledge base is first converted to CNF and then checked for consistency. If the knowledge base is consistent, 0 is returned correspondingly. If it is not, for each proposition x, each clause containing x is removed and the resulting knowledge base is checked for consistency again. This is equivalent to setting x to B in three-valued logic. If one of the new knowledge bases is consistent, 1 is returned correspondingly. If, again, none of the knowledge bases turned out to be consistent, two propositions are set to B, i. e., all possible pairs of propositions are iteratively removed, then all triples, and so forth.

2.2 Answer Set Programming

Answer Set Programming (ASP) [3] is a declarative programming paradigm based on logic programming which is targeted at difficult search problems. ASP incorporates features of Reiter's default logic [12] and logic programming.

[1] http://tweetyproject.org/api/1.14/net/sf/tweety/logics/pl/analysis/ContensionInconsistencyMeasure.html.

An *extended logic program* incorporates both negation-as-failure (not) and classical negation (¬). Such a program comprises rules of the form

$$H \leftarrow A_1, \ldots, A_n, \texttt{not}\, B_1, \ldots, \texttt{not}\, B_m. \tag{1}$$

where H, as well as A_i, $i \in \{1, \ldots, n\}$ and B_j, $j \in \{1, \ldots, m\}$ are literals. In (1), $\{H\}$ is called the *head* of the rule, and $\{A_1, \ldots, A_n, B_1, \ldots, B_m\}$ is called the *body* of the rule. We refer to a set of literals X as *closed under* a positive program P, i. e., a program that contains no instance of not, if and only if for any rule $r \in P$, the head of r is contained in X whenever the body of r is a subset of X. The smallest of such sets wrt. a positive program P is denoted as $\mathsf{Cn}(P)$, and it is always uniquely defined. For an arbitrary program P, a set X is called an *answer set* of P if $X = \mathsf{Cn}(P^X)$ where $P^X = \{H \leftarrow A_1, \ldots, A_n \mid H \leftarrow A_1, \ldots, A_n, \texttt{not}\, B_1, \ldots, \texttt{not}\, B_m. \in P, \{B_1, \ldots, B_m\} \cap X = \emptyset\}$.

A rule with an empty body is referred to as a *fact*, a rule with an empty head is a *constraint*. It should be noted that the head of a rule does not necessarily consist of a single literal – some dialects of ASP allow for constructions such as a *choice rule*, a rule where the head comprises a set of literals of which basically any subset can be set to true. There is also the notion of *cardinality constraints* with lower and upper bounds. Such rules are of the form

$$l\{A_1, \ldots, A_n, \texttt{not}\, B_1, \ldots, \texttt{not}\, B_m\}u \tag{2}$$

The intuition behind this is that a cardinality constraint is only satisfied by an answer set if at least l and at most u of the literals $A_1, \ldots, A_n, B_1, \ldots, B_m$ are included in the answer set. Cardinality constraints can be used as body elements as well as heads of rules [5].

3 Measuring Contension Inconsistency Using ASP

In order to utilise ASP for measuring $\mathcal{I}_c$, i. e., to compute $\textsc{Value}_{\mathcal{I}_c}$, wrt. a knowledge base $\mathcal{K}$, we will encode the problem $\textsc{Upper}_{\mathcal{I}_c}$ in ASP and then send calls to an ASP solver in an iterative manner. By using binary search on the search space of possible inconsistency values, only logarithmic many calls are required. More precisely, wrt. the contension inconsistency measure, the maximum inconsistency value corresponds to the number of atoms n. Thus, the starting point of the binary search is $n/2$.

As a first step in encoding $\textsc{Upper}_{\mathcal{I}_c}$, we create three new propositional atoms e_{x_T}, e_{x_B}, e_{x_F} for each atom x. Thus, the new atoms form a representation of the evaluation of the atom in three-valued logic. For the "guess" part of the ASP, at most u atoms $e_{x_B^i}$, $i \in \{1, \ldots, n\}$ are set to true. This can be modeled as a rule consisting of a cardinality constraint (as introduced in (2)): $0\{e_{x_B^1}, \ldots, e_{x_B^n}\}u$. where u is the upper bound we want to show.

For the "check" part of the ASP, we first need to model that for an atom x only one of its corresponding atoms e_{x_T}, e_{x_B}, e_{x_F} can be evaluated to true:

$$e_{x_T} \leftarrow \texttt{not}\, e_{x_B}, \texttt{not}\, e_{x_F}.,$$
$$e_{x_B} \leftarrow \texttt{not}\, e_{x_T}, \texttt{not}\, e_{x_F}.,$$
$$e_{x_F} \leftarrow \texttt{not}\, e_{x_B}, \texttt{not}\, e_{x_T}.$$

The formulas in $\mathcal{K}$ are comprised of the set of atoms At as well as the operators $\wedge$, $\vee$, and $\neg$. Hence, each operator must be encoded in ASP as well. More specifically, we construct rules that model the evaluation of the formulas $x \wedge y$, $x \vee y$, and $\neg x$ as follows (with new symbols $e_{\ldots}$):

$$x \wedge y \mapsto \quad e_{x\wedge y_T} \leftarrow e_{x_T}, e_{y_T}., \quad e_{x\wedge y_F} \leftarrow e_{x_F}.,$$
$$e_{x\wedge y_F} \leftarrow e_{y_F}., \quad e_{x\wedge y_B} \leftarrow \texttt{not}\, e_{x\wedge y_F}, \texttt{not}\, e_{x\wedge y_T}.$$

$$x \vee y \mapsto \quad e_{x\vee y_F} \leftarrow e_{x_F}, e_{y_F}., \quad e_{x\vee y_T} \leftarrow e_{x_T}.,$$
$$e_{x\vee y_T} \leftarrow e_{y_T}., \quad e_{x\vee y_B} \leftarrow \texttt{not}\, e_{x\vee y_F}, \texttt{not}\, e_{x\vee y_T}.$$

$$\neg x \mapsto \quad e_{\neg x_B} \leftarrow e_{x_B}., \quad e_{\neg x_T} \leftarrow e_{x_F}.,$$
$$e_{\neg x_F} \leftarrow e_{x_T}.$$

More complex formulas can be reduced to these rules. Finally, we need to ensure that all formulas are evaluated either T or B. To achieve this, we add the integrity constraint $\leftarrow e_{\phi_F}.$ for each formula ϕ.

Example 2. We continue Example 1. The ASP corresponding to $\mathcal{K}_1$ would contain the following rules:

1. Cardinality constraint: $0\{e_{a_B}, e_{b_B}, e_{c_B}\}2$. Here, we use 2 as the upper bound.
2. Ensure that each atom only gets one evaluation:

$$e_{a_T} \leftarrow \texttt{not}\, e_{a_B}, \texttt{not}\, e_{a_F}.,\ e_{a_B} \leftarrow \texttt{not}\, e_{a_T}, \texttt{not}\, e_{a_F}.,\ e_{a_F} \leftarrow \texttt{not}\, e_{a_B}, \texttt{not}\, e_{a_T}.,$$
$$e_{b_T} \leftarrow \texttt{not}\, e_{b_B}, \texttt{not}\, e_{b_F}.,\ e_{b_B} \leftarrow \texttt{not}\, e_{b_T}, \texttt{not}\, e_{b_F}.,\ e_{b_F} \leftarrow \texttt{not}\, e_{b_B}, \texttt{not}\, e_{b_T}.,$$
$$e_{c_T} \leftarrow \texttt{not}\, e_{c_B}, \texttt{not}\, e_{c_F}.,\ e_{c_B} \leftarrow \texttt{not}\, e_{c_T}, \texttt{not}\, e_{c_F}.,\ e_{c_F} \leftarrow \texttt{not}\, e_{c_B}, \texttt{not}\, e_{c_T}.$$

3. Encodings for all formulas:
   (a) $a \wedge b$:

$$e_{a\wedge b_T} \leftarrow e_{a_T}, e_{b_T}.,\ e_{a\wedge b_F} \leftarrow e_{a_F}.,\ e_{a\wedge b_F} \leftarrow e_{b_F}.,$$
$$e_{a\wedge b_B} \leftarrow \texttt{not}\, e_{a\wedge b_F}, \texttt{not}\, e_{a\wedge b_T}.$$

   (b) $\neg a \vee b$:

$$e_{\neg a\vee b_F} \leftarrow e_{\neg a_F}, e_{b_F}.,\ e_{\neg a\vee b_T} \leftarrow e_{\neg a_T}.,\ e_{\neg a\vee b_T} \leftarrow e_{b_T}.,$$
$$e_{\neg a\vee b_B} \leftarrow \texttt{not}\, e_{\neg a\vee b_F}, \texttt{not}\, e_{\neg a\vee b_T}.$$

   (c) $\neg b \wedge \neg c$:

$$e_{\neg b\wedge\neg c_T} \leftarrow e_{\neg b_T}, e_{\neg c_T}.,\ e_{\neg b\wedge\neg c_F} \leftarrow e_{\neg b_F}.,\ e_{\neg b\wedge\neg c_F} \leftarrow e_{\neg c_F}.,$$
$$e_{\neg b\wedge\neg c_B} \leftarrow \texttt{not}\, e_{\neg b\wedge\neg c_F}, \texttt{not}\, e_{\neg b\wedge\neg c_T}.$$

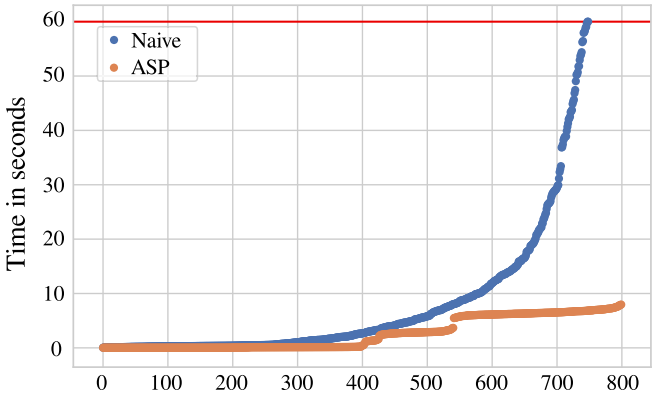

Fig. 1. Comparison of the (naive) state of the art implementation of the contension inconsistency measure and the (ASP) implementation of the algorithm proposed in this work. The red horizontal line visualises the timeout of 60 s. (Color figure online)

(d) Negations:

$$e_{\neg a_B} \leftarrow e_{a_B}., \; e_{\neg a_T} \leftarrow e_{a_F}., \; e_{\neg a_F} \leftarrow e_{a_T}.,$$
$$e_{\neg b_B} \leftarrow e_{b_B}., \; e_{\neg b_T} \leftarrow e_{b_F}., \; e_{\neg b_F} \leftarrow e_{b_T}.,$$
$$e_{\neg c_B} \leftarrow e_{c_B}., \; e_{\neg c_T} \leftarrow e_{c_F}., \; e_{\neg c_F} \leftarrow e_{c_T}.$$

4. Integrity constraints: $\leftarrow e_{a \wedge b_F}., \leftarrow e_{\neg a \vee b_F}., \leftarrow e_{\neg b \wedge \neg c_F}.$

4 Preliminary Experiments

The algorithm presented in the previous section is implemented in Java by use of the TweetyProject[2] library. The library already provides an implementation of the contension inconsistency measure that constitutes the state of the art.

In order to evaluate the proposed ASP algorithm, we compare its implementation with the naive one. We created a total of 800 random knowledge bases of different sizes and complexities. The knowledge bases are comprised of around 15-20 formulas which contain 0-10 connectors. To achieve this, we utilised a sampler (namely, the `SyntacticRandomSampler`[3]) provided by the TweetyProject. The generated knowledge bases are built on signatures that contain either 5, 10, or 15 propositional atoms. Then we applied both algorithms on each of these knowledge bases and measured the execution time. A timeout was set to 60 s. Figure 1 displays the measured execution time regarding each knowledge base, sorted from low to high wrt. both algorithms. Clearly, the ASP algorithm performs more efficiently. While applying the naive algorithm produced a timeout in 53 cases, the ASP implementation required only a maximum of 7.97 s to return the inconsistency value.

[2] http://tweetyproject.org/index.html.
[3] http://tweetyproject.org/api/1.14/net/sf/tweety/logics/pl/util/SyntacticRandomSampler.html.

5 Conclusion

In this paper, we introduced an algorithm for calculating the contension inconsistency measure by means of reductions to answer set programming. By providing rules for encoding three-valued evaluations of propositional formulas in ASP rules, an inconsistency value can be retrieved using only logarithmic many calls to an answer set solver. In Sect. 4 we compared an implementation of a state of the art algorithm for calculating contension inconsistency with the proposed method. The evaluation shows that the ASP algorithm clearly outperforms the state of the art. This quite positive result leads to the conclusion that reductions to ASP are a reasonable method to approach problems in the field of inconsistency measurement. Consequently, it would be useful to explore the calculation of other inconsistency measures using reductions to ASP as well.

References

1. Atkinson, K., et al.: Toward artificial argumentation. AI Mag. **38**(3), 25–36 (2017)
2. Béziau, J.Y., Carnielli, W., Gabbay, D. (eds.): Handbook of Paraconsistency. College Publications, London (2007)
3. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
4. De Bona, G., Grant, J., Hunter, A., Konieczny, S.: Towards a unified framework for syntactic inconsistency measures. In: Proceedings of the AAAI 2018 (2018)
5. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer set solving in practice. Synth. Lect. Artif. Intell. Mach. Learn. **6**(3), 1–238 (2012)
6. Grant, J.: Classifications for inconsistent theories. Notre Dame J. Formal Logic **19**(3), 435–444 (1978)
7. Grant, J., Hunter, A.: Measuring consistency gain and information loss in stepwise inconsistency resolution. In: Liu, W. (ed.) ECSQARU 2011. LNCS (LNAI), vol. 6717, pp. 362–373. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22152-1_31
8. Grant, J., Martinez, M. (eds.): Measuring Inconsistency in Information. College Publications, London (2018)
9. Hansson, S.: A Textbook of Belief Dynamics. Kluwer Academic Publishers (2001)
10. Hunter, A., Konieczny, S.: Measuring inconsistency through minimal inconsistent sets. In: Proceedings of the KR 2008, pp. 358–366 (2008)
11. Priest, G.: Logic of paradox. J. Philos. Logic **8**, 219–241 (1979)
12. Reiter, R.: A logic for default reasoning. Artif. Intell. **13**(1–2), 81–132 (1980)
13. Thimm, M.: Measuring inconsistency with many-valued logics. Int. J. Approximate Reasoning **86**, 1–23 (2017)
14. Thimm, M.: On the evaluation of inconsistency measures. In: Measuring Inconsistency in Information. College Publications (2018)
15. Thimm, M., Wallner, J.: On the complexity of inconsistency measurement. Artif. Intell. **275**, 411–456 (2019)