Reasoning About Inconsistent Formulas

Joao Marques-Silva\(^1\) and Carlos Mencia\(^2\)
\(^{1}\)ANITI, University of Toulouse, France
\(^{2}\)University of Oviedo, Spain

Abstract

The analysis of inconsistent formulas finds an ever-increasing range of applications, that include axiom pinpointing in description logics, fault localization in software, model-based diagnosis, optimization problems, but also explainability of machine learning models. This paper overviews approaches for analyzing inconsistent formulas, focusing on finding and enumerating explanations of and corrections for inconsistency, but also on solving optimization problems modeled as inconsistent formulas.

1 Introduction

In a wide range of settings, one must reason about inconsistent formulas. This is the case for example when debugging inconsistent system specifications, e.g. for understanding which constraints make the system inconsistent. This is also the case in model-based diagnosis [Reiter, 1987], where the system representation is inconsistent with its expected input-output behavior. In turn, model-based diagnosis finds a number of significant applications, that include axiom pinpointing in description logics, software fault localization, type error debugging, spreadsheet debugging, or analysis of inconsistent knowledge bases, to name a few. Inconsistency is also often used when solving optimization problems, e.g. in the case of maximum satisfiability for propositional domains.

In this paper we focus on inconsistent formulas of monotonic logics, e.g. first-order logic and its fragments. However, there is also interest in non-monotonic logics [Brewka et al., 2019]. When analyzing inconsistent formulas, a number of computational problems arise. In some settings, one is interested in explaining one possible reason of inconsistency. In some other settings, the goal is to propose ways to correct the source of inconsistency. Finally, in yet some other settings, the purpose is to enumerate all explanations or corrections of an inconsistent formula.

The purpose of this paper is to provide an overview of recent work on reasoning about inconsistent formulas. The paper addresses the problems of explaining and correcting inconsistency, but also highlights enumeration of explanations and corrections, and briefly overviews how inconsistency is exploited when solving optimization problems. The paper also demonstrates that most of the algorithms proposed in recent years for reasoning about inconsistency, most of which originally devised in the context of analyzing propositional formulas, can be also applied in any monotonic logic.

The paper is organized as follows. Section 2 introduces the definitions and notation used throughout the paper. Section 3 covers approaches for finding (or extracting) subset-minimal explanations and corrections. Section 4 overviews the relationship between reasoning about inconsistency and optimization problems. Section 5 highlights algorithms for the enumeration of both corrections and explanations. A sample of practical applications is briefly discussed in Section 6. Finally, the paper concludes in Section 7.

2 Preliminaries

Basic definitions. The paper assumes definitions standard for first-order logic (FOL) [Ben-Ari, 2012]. We will be interested in formulas expressed in some (decidable) fragment of FOL\(^1\), and so respecting monotonicity of entailment. For simplicity, we will not consider the use of function symbols, but that does not affect the results in the paper. Throughout, FOL formulas will be represented in clausal form and will most often be inconsistent. In general, we consider an inconsistent FOL formula \(F\), such that some clauses in \(F\) can be relaxed (i.e. allowed not to be satisfied) to restore consistency, whereas others cannot. Thus, we assume that \(F\) is partitioned into two first-order subformulas \(F = B \cup S\), where \(S\) contains the relaxable (soft) clauses, and \(B\) contains the non-relaxable clauses. \(B\) can be viewed as background knowledge, which is assumed to be consistent and must always be satisfied.

The paper uses other standard notation. \(P \models Q\) denotes that any model of \(P\) is also a model of \(Q\). As a result, \(P \models \bot\) is used to denote that \(P\) is inconsistent, and \(P \models \bot\) denotes that \(P\) is consistent. Set notation will be used to improve readability. For example, \(P \cup Q \models R\) is used to denote \(\bigwedge_{c \in P \cup Q} c \models R\).

Generalizations from the propositional domain. This paper covers the analysis of inconsistent formulas in very generic terms, opting not to focus exclusively on propositional formulas. A number of concepts often used in the

\(^1\)As a result, the results presented in the paper apply in different settings, including SAT, ILP, CSP, QBF, SMT (specific theories), among others. Decidability is expected, since we will need to be able to answer decision problems both positively and negatively.
propositional domain can be easily generalized to different fragments of first-order logic. These will prove instrumental in showing how different algorithms, often described for propositional formulas, can in fact be generalized well beyond. A simple observation is that propositional variables can be viewed as predicate symbols. Thus, concepts such as refutation of clauses, or constraints over propositional variables, can be used in more general settings. For example, pseudo-boolean constraints, including cardinality constraints, can be readily used in fragments of FOL. Furthermore, although QBF may seem to involve non-standard quantification (from a FOL perspective), there are well-known translations from QBF to EPR [Seidl et al., 2012], and so the proposed generalizations also extend to QBF.

Subset-minimal explanations and corrections. Given an inconsistent clausal formula $F$, we are interested in identifying the clauses that are responsible for unsatisfiability among those that can be relaxed, as defined next.

**Definition 1 (MUS).** Let $F = B \cup S$ be an inconsistent set of clauses ($F \models \bot$), $M \subseteq S$ is a Minimal Unsatisfiable Subset (MUS) iff $B \cup M \models \bot$ and $\forall M' \subseteq M, B \cup M' \not\models \bot$.

Informally, an MUS provides the minimal information that needs to be added to the background knowledge $B$ to obtain inconsistency; thus, an MUS represents an explanation for the causes of inconsistency. Alternatively, one might be interested in correcting the formula, removing some clauses to achieve consistency.

**Definition 2 (MCS).** Let $F = B \cup S$ be an inconsistent set of clauses ($F \models \bot$), $C \subseteq S$ is a Minimal Correction Subset (MCS) iff $B \cup S \setminus C \not\models \bot$ and $\forall C' \subseteq C, B \cup S \setminus C' \not\models \bot$.

With each MCS $C$, one associates a Maximal Satisfiable Subset (MSS), given by $S \setminus C$. Moreover, there exists a well-known (subset-)minimal hitting set (MHS) relationship between MUSes and MCSes.

**Proposition 1.** MCSes are MHSes of MUSes and vice-versa.

The MHS relationship between MUSes and MCSes was first demonstrated in the context of model-based diagnosis [Reiter, 1987] and later investigated for propositional formulas in clausal form [Birnbaum and Lozinskii, 2003].

Recent years have witnessed the proposal of a large number of novel algorithms for the extraction and enumeration of MUSes and MCSes [Mencía et al., 2015; Bacchus and Katsirelos, 2015; Lifitton et al., 2016; Bacchus and Katsirelos, 2016; Mencía et al., 2016; Previti et al., 2018; Grégoire et al., 2018; Narodytska et al., 2018; Bendik et al., 2018], where MUS enumeration algorithms build on the MHS relationship between MUSes and MCSes.

Tractable theories have also been studied, as Horn formulas [Marques-Silva et al., 2016]. In addition, there has been interest in computing the union of all MUSes [Mencía et al., 2019], which can be approximated by the so-called lean kernel [Kleine Büning and Kullmann, 2009; Kullmann and Marques-Silva, 2015; Peñaloza et al., 2017].

Minimal sets over monotone predicates (MSMP). The similarity among solutions for a number of apparently unrelated problems motivated the proposal of a unifying paradigm, MSMP [Marques-Silva et al., 2013b; Marques-Silva et al., 2017], consisting in computing minimal sets over a monotone predicate. In this setting, a predicate $P: 2^{R} \rightarrow \{0, 1\}$, defined over a reference set $R$, is monotone if whenever $P(R_o)$ holds, with $R_0 \subseteq R$, then $P(R_1)$ also holds, with $R_0 \subseteq R \subseteq R_1$. By defining different problems as instances of MSMP, one can devise algorithms that can be used to solve different problems, many of which are related with the analysis of inconsistent formulas. Earlier work [Marques-Silva et al., 2017] proposed a partition of MSMP problems into three classes, depending on the type of monotone predicate considered. (Due to space constraints, the predicate forms are only hinted at in this paper.)

**Maximum satisfiability and optimization problems.** Given $F = B \cup S$, with $B \cup S \not\models \bot$, the maximum satisfiability (MaxSAT) problem asks for the largest subset $S'$ of $S$, such that $B \cup S' \not\models \bot$. A cost can be associated with each clause in $S$, and so the MaxSAT problem corresponds instead to selecting the subset $S'$ of the largest cost. There is a well-known relationship between MaxSAT solutions and MSSes, in that every MaxSAT solution is an MSS, and so the complement of a MaxSAT solution is an MCS. Furthermore, it is well-known that standard optimization problems can be cast as MaxSAT (where MaxSAT can be interpreted at the propositional level, but also within any fragment of FOL): $\min_{S} \sum_{i} w_{i} \times p_{i}$ (1) where $w_0 \geq 0$, and each $p_i$ can be viewed as a 0-place predicate. In this case, just let $B = M$, $S = \{\neg p_{i}\}$ and associate with each clause $c_i$ in $S$ a cost $w_i$. Thus, $F = B \cup S$ is inconsistent and an MSS (in terms of cost) of $F$ is a solution to the optimization problem.

**Running example.** Figure 1 shows an example used throughout this paper to illustrate the operation of algorithms for extracting and enumerating MUSes and MCSes. As it can be observed, the example is written in Bernays-Schönfinkel’s fragment of first-order logic, also referred to as effectively propositional logic (EPR). It is well-known that EPR is NEXP-complete [Lewis, 1980]. In this example, we have a knowledge base (see Figure 1a) with a number of general statements about parents ($P$) of new newborn children ($N$) being proud ($R$), and that fathers or mothers are parents. We also indicate that there should be no proud people (with the objective of finding proud parents). Moreover, we also have statements about concrete people $(\{aw, rw, hw, pw, mw\})$. We declare the general statements as hard, denoting that we are certain that the statements are true. In contrast, the information about concrete people is declared as soft, denoting that we are uncertain about whether the information is true or not. The set of statements is inconsistent. Using an existing reasoner for EPR [Korovin, 2013] as an oracle, we can identify

2Adapted from [Nilsson and Maluszynski, 1995, Section 3.1].

3By using EPR, we highlight that most of the proposed algorithms generalize beyond the propositional cases and beyond oracles for NP. There are dedicated algorithms for the analysis of EPR formulas [Xie and Luo, 2016], but these are fundamentally different from the algorithms studied here.
two minimal explanations (MUSes) of inconsistency as well as four minimal corrections (MCSes) for eliminating inconsistency. These sets are shown in Figure 1b. The justification for the inconsistencies serve to convey the facts that aw and rw can be inferred to be proud parents, and this is inconsistent with the requirement of no one being proud. As can be observed, the MUSes are MHSes of the MCSes and vice-versa.

3 Subset-Minimal Sets

This section overviews approaches for extracting subset-minimal explanations and corrections of inconsistency.

Subset-minimal explanations. Given an inconsistent formula $F = B \cup S$, there is a fairly straightforward algorithm for finding one MUS. Starting from a set of clauses $M \models S$, iteratively remove one clause $u$ from $M$. If $B \cup (M - \{u\})$ is inconsistent, then we can safely discard $u$. Otherwise, we must keep $u$, to preserve inconsistency. After analyzing all clauses of $S$, the resulting set $M$ is an MUS of $F$. Variants of this algorithm have been studied since at least the early 1990s [Chinneck and Dravnieks, 1991; Bakker et al., 1993; Dershovitz et al., 2006]. Algorithm 1 represents a possible instantiation of this approach for computing MUSes, but which also exploits the MSMP framework. As a result, one must specify which predicate to use for MUS extraction. Following [Marques-Silva et al., 2013b; Marques-Silva et al., 2017], we use the predicate $P(B, S, W) \models \neg \text{SAT}(B \wedge W)$, with $W \subseteq R$, and $R \models S$. The SAT test for the running example needs to decide the satisfiability of an EPR formula. In general, SAT must solve the decision problem for the logic used to represent $B$ and $R$. Figure 2 summarizes a possible execution of Algorithm 1 on the formula from Figure 1. Clearly, the computed MUS depends on the order of clauses considered. In this case, the computed MUS is $\{c_3, c_4\} \models \{F(aw, pw), N(pw)\}$ which, together with $B$, is inconsistent and minimal.

It is simple to conclude that Algorithm 1 requires $\Theta(|R|)$ predicate tests. There is no known approach for computing one MUS that improves asymptotically on this bound. A number of algorithms for MUS extraction have been proposed over the years e.g. [Junker, 2004; Marques-Silva et al., 2013b] but without exception, in the worst-case, the number of predicate calls is $\Omega(|S|)$. Depending on the target decision problem, different optimizations can be envisioned for MUS extraction, e.g. [Belov et al., 2012; Wieringa, 2012; Lonsing and Egly, 2015]. Table 1 summarizes the query complexity of well-known algorithms for extracting a single MUS, where $m$ and $k$ denote the number of clauses in $S$ and the size of the largest minimal set respectively.

Subset-minimal corrections. Given an inconsistent formula $F = B \cup S$, the following approach can be used for finding one MCS. Starting from a set of clauses $M \models 0$, iteratively consider one clause $u \in S$. If $B \cup M \cup \{u\}$ is consistent, add $u$ to $M$. After considering all clauses in $S$, the clauses that could not be added to $M$ denote an MCS of $F$. Such an algorithm has been used since at least the mid 00s [Bailey and Stuckey, 2005], and has more recently
been referred to as basic linear search (BLS) [Marques-Silva et al., 2013a]. As implicit in earlier work [Marques-Silva et al., 2013b; Marques-Silva et al., 2017], by changing the predicate used, Algorithm 1 corresponds to BLS, as outlined above. For this case, the predicate to use is defined as follows: $P(B, S, W) = \text{SAT}(B \land (S \setminus W))$, with $W \subseteq R$ and $R \triangleq S$. Figure 3 illustrates the execution of Algorithm 1 for computing an MCS. The computed set is $\{c_2, c_4\}$. A number of optimizations to the BLS algorithm have been proposed in earlier work [Marques-Silva et al., 2013a; Mencía et al., 2015]. These include, among others, exploiting models to save a number of predicate tests.

BLS is one of a number of algorithms that have been proposed for extracting MCSES. A conceptually simple solution is to find an MCS using maximum satisfiability [Liffiton and Sakallah, 2008]. In practice, solving MaxSAT, even if requiring fewer oracle calls in the worst case, is not as effective as other alternatives. One of the most widely used methods for the extraction of MCSES is the Clause D (CLD) algorithm [Marques-Silva et al., 2013a]. For some kinds of MSMP problems, the CLD algorithm can be used.

Algorithm 2 summarizes the main steps of CLD. One starts by reifying each clause $c_i \in S$ with a fresh relaxation variable (or 0-place predicate): $(r_i \lor \neg c_i)$, thus creating a replacement $R'$ for the original $R$. Moreover, a disjunction is created with a literal $\neg r_i$ for each $r_i$ variable (this is the D clause). Then, the algorithm iteratively tests $B \lor R' \lor \{D\}$. (Observe that this requires at least one $r_i$ to be assigned value 0.) If this formula is satisfiable, then some soft clause(s) $c_i$ can be satisfied without assigning $r_i$ to 1. Thus, we recreate the formula by dropping any $r_i$ assigned value 0 and recreate the D clause without the corresponding literals. The process is repeated until the D clause contains a set of literals that, given the other clauses, cannot be satisfied. These literals identify an MCS$^5$. In contrast to the deletion-based approach, as well as to many other algorithms for MUS extraction, CLD expects the oracle call to return a model, from which it can identify the clauses that will no longer be reified. Thus, the predicate call must test for consistency and not for inconsistency. Nevertheless, as shown in earlier work, a large number of minimal set problems are defined by testing for consistency.

Table 2 summarizes the query complexity of well-known algorithms for extracting a single MCS, where $n$ is the number of variables in $S$, and $m$ and $k$ have the same meaning as before, with the exception of CLD, where $k$ is the size of the smallest MCS. Moreover, there are algorithms that compute an MCS by modifying the oracle, imposing a fixed branching preference [Rosa et al., 2010; Bacchus et al., 2014].

### 4 Cardinality- & Preferred Minimal Sets

#### Solving maximum satisfiability

MaxSAT is solved by finding smallest (cost) MCSES. A well-known approach for solving MaxSAT is branch-and-bound search (B&B) [Li and Manyà, 2009]. Since the mid 00s, experimental evidence from a number of applications showed that B&B does not scale in practice [Morgado et al., 2013; Ansótegui et al., 2013]. An alternative approach consists in reifying all the soft clauses, i.e. replace $(c_i)$, with $c_i \in S$, with $(r_i \lor \neg c_i)$, obtaining $S_r$; and then defining a constraint $X \triangleq \sum_i w_i \times r_i \leq \tau$, where $\tau$ ranges from 0 to $\sum_i w_i$. As a result, finding the smallest value of $\tau$, such that $B \land S_r \land X$ is consistent, can then be achieved with linear search [Le Berre and Perrain, 2010], by refining lower or upper bounds on the value of $\tau$, or with binary search [Koshimura et al., 2012]. When the number of soft clauses is large, the representation or handling of $X$ can be a problem. Motivated by this drawback, two approaches for solving MaxSAT using SAT oracles have been proposed since the mid 00s, both of which reason about inconsistency. Core-guided approaches relax soft clauses.

---

$^5$The propositional version admits optimizations, including the fact that reification is unnecessary. Some optimizations are not restricted to the propositional version.

$^6$In the propositional case one obtains $O(\min(n, m - k))$.

$^7$LBX can only be used with non-quantified formulas.
on demand (see [Morgado et al., 2013; Ansótegui et al., 2013] for surveys of earlier work and [Martins et al., 2014; Morgado et al., 2014; Alviano et al., 2015] for recent improvements). An alternative is based on the iterative computation of minimum hitting sets (MinHS) on clauses describing (minimal) unsatisfiable cores [Davies and Bacchus, 2011]. Core-guided approaches reify clauses on demand given identified (minimal) unsatisfiable subsets. Cardinality (for unweighted problems) or pseudo-boolean constraints (for weighted problems) are then specified using the relaxation variables and existing lower or upper bounds. MinHS approaches do not reify clauses on demand. Instead, computed (minimal) unsatisfiable subsets serve as sets that must be hit when selecting the next minimum cost subset of clauses whose satisfiability is to be tested. Not surprisingly, both approaches generalize beyond the propositional case.

Smallest explanations. The previous section argued that existing algorithms for computing an MUS require \(\Theta(|S|)\) calls to a suitable oracle. In contrast, computing a cardinality-minimal (or smallest) MUS (SMUS) seems to require solving a computationally harder problem. For the case of propositional logic, deciding whether there exists an MUS of size not greater than some \(k\) is complete for \(\Sigma_2^p\) [Liberatore, 2005], and so computing a smallest MUS requires a logarithmic number of calls to a \(\Sigma_2^p\) oracle. The state of the art in computing smallest explanations is FORQES [Ignatiev et al., 2015]. As with other methods studied in this paper, the FORQES algorithm generalizes beyond the propositional case. Earlier work proposed worst-case exponential propositional encodings for this problem [Lyne and Marques-Silva, 2004].

Computing preferred sets. In propositional logic, preferences of inclusion of clauses in MUSes and MCSes are hard for the second level of the polynomial hierarchy [Marques-Silva and Previdi, 2014]. Preferences of non-inclusion of clauses in MUSes and MCSes are in \(FP^{NP}\). Similar increases in complexity are expected to hold beyond the propositional case.

5 Enumeration of Minimal Sets

Enumeration of minimal corrections. Enumeration of MCSes can be achieved by enumerating MaxSAT solutions [Liffton and Sakallah, 2008]. More recent work proposed dedicated algorithms for MCS enumeration [Marques-Silva et al., 2013a; Bacchus et al., 2014; Previdi et al., 2017; Previdi et al., 2018; Grégoire et al., 2018; Narodytska et al., 2018]. To implement enumeration of MCSes, one needs to prevent the same MCS from being computed again. This can be done by blocking the selection of the set of clauses corresponding to an MCS. Most recent work focuses on how to optimize the enumeration process.

Enumeration of minimal explanations. To our best knowledge, enumeration of minimal explanations is harder than the enumeration of minimal corrections, and cannot be obtained directly by iterative extraction of MUSes\(^8\). One well-known solution is the enumeration of all MCSes, from which all the MUSes can be obtained by computing all the minimal hitting sets. This was implemented for example in the CAMUS tool [Liffton and Sakallah, 2008]. A difficulty of these approaches is that MUSes can be computed only after all the MCSes have been computed, which can be exponential in number. A recent alternative is the MARCO algorithm [Liffton et al., 2016], which iteratively enumerates both MUSes and MCSes, and can be configured such that only one kind of minimal set needs to be extracted.

Algorithm 3 outlines a modified MARCO algorithm, where the preference for a specific kind of minimal set is made explicit. The two configurations are shown in Table 3. Let us focus on the one that gives preference to finding MUSes (the other one is analogous). In this case, XYZ is set to MUS, and ABC to MCS. The algorithm maintains two sets of sets of indices (where index \(i_j\) is associated to clause \(c_j \in S\); \(X\)), with sets blocking all MUSes found; and \(P\), with sets blocking all MCSes found. Iteratively, a minimal hitting set (MHS) \(\lambda\) of \(X\) subject to \(P\) is computed, which induces the set of clauses \(\text{cls}(\lambda)\). Then, if \(\neg\text{SAT}(B \land S \setminus \text{cls}(\lambda))\) holds, an MUS is extracted from \(S \setminus \text{cls}(\lambda)\), which is blocked in \(X\). Otherwise, \(\text{cls}(\lambda)\) represents an MCS, which is blocked in \(P\). The process continues until no MHS exists, guaranteeing that all MUSes and MCSes have been reported. Figure 4 illustrates the execution of Algorithm 3 on the running example. Improvements to MARCO have been proposed in recent years [Bacchus and Katsirelos, 2016; Narodytska et al., 2018; Bendik et al., 2018]. Besides, MARCO and variants have a growing number of applications [Polikarpova et al., 2016; Rothenberg and Grumberg, 2016; Brandt et al., 2018].

---

\(^8\)Implicit enumeration of all possible sets [Reiter, 1987; Bailey and Stuckey, 2005] could be considered, but these approaches do not scale in practice [Liffton and Sakallah, 2008].

Algorithm 3: Generic MUS/MCS enumeration

**Function** \(\text{MINSETENUM}(P_{XYZ}, B, S)\)

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((\mathcal{N}, \mathcal{P}) \leftarrow (\emptyset, \emptyset);)</td>
</tr>
<tr>
<td>2</td>
<td>while (\text{true}) do</td>
</tr>
<tr>
<td>3</td>
<td>((\sigma_{\lambda}, \lambda) \leftarrow \text{FindMHS}(\mathcal{N}, \mathcal{P});)</td>
</tr>
<tr>
<td>4</td>
<td>if (\neg \sigma_{\lambda}) then break;</td>
</tr>
<tr>
<td>5</td>
<td>((\sigma_{\mu}, \mu) \leftarrow \text{ChkXYZ}(B, S, \text{cls}(\lambda));)</td>
</tr>
<tr>
<td>6</td>
<td>if (\sigma_{\mu}) then</td>
</tr>
<tr>
<td>7</td>
<td>(\tau \leftarrow \text{GetXYYZ}(P_{XYZ}, B, S, \sigma_{\mu}, \text{cls}(\lambda));)</td>
</tr>
<tr>
<td>8</td>
<td>ReportXYZ((\tau));</td>
</tr>
<tr>
<td>9</td>
<td>(\mathcal{N} \leftarrow \mathcal{N} \cup \text{NegClIDs}(\tau));</td>
</tr>
<tr>
<td>10</td>
<td>else</td>
</tr>
<tr>
<td>11</td>
<td>ReportABC((\text{cls}(\lambda)));</td>
</tr>
<tr>
<td>12</td>
<td>(\mathcal{P} \leftarrow \mathcal{P} \cup \text{PosClIDs}(\text{cls}(\lambda)));</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC XYZ</td>
<td>ChkXYZ</td>
</tr>
<tr>
<td>MCS MUS</td>
<td>(-\text{SAT}(B \land S \setminus \text{cls}(\lambda)))</td>
</tr>
<tr>
<td>MUS MCS</td>
<td>(-\text{SAT}(B \land \text{cls}(\lambda)))</td>
</tr>
</tbody>
</table>

Table 3: Configuration of Algorithm 3
6 Example Applications

This section provides a brief glimpse of the applications of reasoning about inconsistency. Optimizing linear cost functions subject to sets of constraints is ubiquitous in a growing range of domains. One well-known application domain is program analysis [Si et al., 2017]. Additional examples include Pseudo-Boolean Optimization but also optimization subject to quantified constraints and multi-objective optimization [Terra-Neves et al., 2017]. The relationship between the analysis of inconsistent formulas and model-based diagnosis (MBD) can be traced to the seminal work of Reiter [Reiter, 1987]. Approaches based on solving MaxSAT or computing MCSes have been proposed [Metodi et al., 2014; Marques-Silva et al., 2015; Ignatiev et al., 2019a]. A concrete instantiation of MBD is axiom pinpointing. In this respect, several of the algorithms presented in this paper have been applied to axiom pinpointing in $\mathcal{EL}^+$ ontologies [Sebastiani and Vescovi, 2009; Arif et al., 2015]. In addition, MUS extraction finds application in formal verification and model checking [McMillan and Amla, 2003; Nadal, 2010]. Explainability is arguably a strategic area in Machine Learning (ML). Most existing approaches for explaining ML models are heuristic, with recent results raising concerns about the global validity of the computed explanations [Ignatiev et al., 2019d]. Recent work relates explanations with prime implicants, and so with analyzing inconsistent formulas [Ignatiev et al., 2019b]. Furthermore, recent work has also shown important links between explanations, adversarial examples and the seminal work of Reiter [Ignatiev et al., 2019c].

7 Conclusions & Research Directions

This paper overviews solutions for the analysis of inconsistent formulas, including the identification of minimal corrections and minimal explanations, but also their enumeration. The paper highlights that most of the algorithms proposed for reasoning about inconsistent formulas generalize beyond the propositional case, being applicable in more expressive logics. Motivated by the many existing applications, the recent fast pace of improvement of algorithms for inconsistency analysis is expected to continue in the near future.

Acknowledgements

This work is supported by the AI Interdisciplinary Institute ANITI, funded by the French program “Investing for the Future – PIA3” under Grant agreement n° ANR-19-PIA-0004, by the Spanish Government under project TIN2016-79190-R and by the Principality of Asturias under grant IDI/2018/000176.

References


