Enforcing Logically Weaker Knowledge in Classical Logic

Philippe Besnard[1], Éric Grégoire[2], and Sébastien Ramon[2]

[1] IRIT CNRS UMR 5505
118 route de Narbonne, F-31062 Toulouse, France
besnard@irit.fr
[2] Université Lille - Nord de France, Artois, F-62307 Lens
CRIL, F-62307 Lens
CNRS UMR 8188, F-62307
rue Jean Souvraz SP18, F-62307 Lens France
{gregoire,ramon}@cril.univ-artois.fr

Abstract. This paper is concerned with a fundamental issue in knowledge representation and reasoning that, surprisingly, has received little attention so far. The point is that inserting some logically weaker (but, in a sense, more precise) information within a logic-based representation is not a straightforward process if the extra information must prevail. Indeed, it does neither prevail by itself nor disable the already existing logically stronger (but less precise) information that subsumes it. A general framework for solving this problem is introduced and instantiated to the task of making some rules prevail over more general ones.

1 Introduction

Since the early stage of the Artificial Intelligence (A.I.) research field, it has been recognized that logic can play several roles in knowledge representation and reasoning, either as a competence model [14] at the e.g. so-called knowledge level [15] or as the actual tool for implementation. Recently, there has been a surge of interest in logic-based representations due to a dramatic progress in the size and the actual difficulty of problems that can be handled by automated theorem provers and consistency checking tools based on SAT solvers [3] [1].

This paper is thus concerned with knowledge representation and reasoning within classical logic. These last decades several pitfalls of classical logic have been addressed successfully by the A.I. research community, like e.g. its monotonicity property [2] that appears too restrictive or the way complete deductive systems collapse in the presence of contradictory information [18]. Quite surprisingly, the following question has not received much attention so far from a logic-based point of view, although it is ubiquitous in the dynamics of reasoning and learning.

Assume that we want to enrich a logical knowledge base Γ by some piece of information that is already subsumed by Γ (so that the extra piece information is in a sense more precise than the subbase -of Γ- that subsumes it). For instance, Γ contains the rule '*If the switch is on then the light is on*' and we want Γ to be enriched with the more precise rule '*if the switch is on and if the lamp bulb is ok then the light is*

on.' In the setting of classical logic, the latter rule is already entailed by the first rule. This actually means the following. Whenever the precondition of the first rule holds, that is, '*the switch is on*', it happens that '*the light is on*' (which is the conclusion of that rule) is derived, no matter what extra information is added, like e.g. '*the lamp bulb is not ok*.' This drawback is due to the monotonicity property of standard logic, which makes any derivable conclusion to remain derivable whatever new information comes in. Theory revision and nonmonotonic logics have been devised to address the problem of handling new information that conflicts with the existing one. By contrast, the new piece of information in the above example is *not* contradictory with Γ. Hence, theory revision and nonmonotonic logics are not a way out. What we need is an approach that makes the new piece of information prevail: '*the light is on*" can be derived only when both conditions '*the switch is on*' and '*the lamp bulb is ok*' are themselves derivable.

This issue might seem straightforward at first glance: To solve it, we would simply need to get rid of the older rule and of all other rules that are logically "stronger" than the new one. We would also need to break all reasoning paths (possibly involving various rules) that would yield the pattern encoded by a dismissed rules. Even this needs not be enough as inserting the new rule that must prevail might actually enable or even restore reasoning paths that we would have just expelled. The paper investigates this issue from a classical logic-based account and provides a detailed solution.

The paper is organized as follows. First, some very basic technical background about Boolean representations and logic is recalled. Then, in Section 3, the above issue is expressed more formally. It is shown that the seemingly straightforward solution is wrong. Section 4 presents a solution to the problem. Computational investigations are presented in Section 5. Related works are discussed in Section 6 and promising paths for further research are presented in the conclusion.

2 Logical background

Throughout the paper, the following notations are used: $\neg$, $\vee$, $\wedge$ and $\rightarrow$ denote the classical negation, disjunction, conjunction and material implication connectives, respectively. $\bot$ and $\top$ denote absurdity and any tautology, respectively.

Let Γ be a finite set of Boolean formulas, built in the classical way from the connectives and Boolean variables. An interpretation of Γ assigns values from $\{true, false\}$ to all variables occurring in Γ and sets the truth value of any formula of Γ in the usual compositional way. A model m of Γ is an interpretation that satisfies all formulas in Γ, i.e. makes them $true$. It is represented by the set of literals it satisfies. The set of models of Γ is noted $\mathcal{M}(\Gamma)$. A formula f can be deduced from Γ when f is $true$ in all models of Γ, which is denoted $\Gamma \models f$. $Cn(\Gamma)$ denotes the deductive closure of Γ, i.e. the set of all formulas that can be deduced from Γ. Γ is inconsistent when it has no model: Γ is then equivalent to $\bot$. We will note $f \models g$ and say that a formula g is a logical consequence of a formula f when $\mathcal{M}(g) \subseteq \mathcal{M}(f)$. When $f \models g$ and $g \models f$, g and f are said equivalent, this is noted $f \equiv g$. We note $f \models_\Gamma g$ and say that g is a logical consequence of f modulo a set of formulas Γ when $\Gamma \cup \{f\} \models g$. When $f \models_\Gamma g$ and $g \models_\Gamma f$, f and g are said equivalent modulo Γ, noted $f \equiv_\Gamma g$.

Every Boolean formula can be rewritten in clausal form -equivalent w.r.t. satisfiability-, i.e. as a conjunction of clauses, where a clause is a disjunction of literals, where a literal is a signed Boolean variable, namely a variable x or its negated form $\neg x$. It is convenient to represent a clause by means of the set of its literals. In this respect, subclauses of a given clause f are represented by subsets of f.

We differentiate between *syntax-based* representations which take Γ to be a set of formulas and *semantic-based* approaches to knowledge representation which view Γ as a set of models. In the latter case, the set of formulas corresponding to Γ is deductively closed whereas this is not mandatory in syntax-based approaches.

The problem of checking whether a set of clauses is satisfiable and of exhibiting a model of it in the positive case is called SAT. SAT is NP-complete. A minimal unsatisfiable subset (in short, MUS) Σ of a set Γ of formulas is a subset of Γ that is unsatisfiable and that is such that any proper subset of Σ is satisfiable.

3 Logical characterization

Let g be a rule that is meant to prevail when inserted within a set Γ of formulas, in the sense that any possibility to derive any rule f that is "stronger" than g must be disabled. In the example from the introduction, g is an encoding of '*if the switch is on and if the lamp bulb is ok then the light is on*' whereas one specific f is an encoding of '*if the switch is on then the light is on*.' In the following, we assume that f and g are clauses that are neither tautological nor inconsistent, and that Γ and Γ' are two sets of formulas.

From a technical point of view, we need to transform Γ into Γ' s.t. Γ' allows us to derive g but does not enable us to derive any clause f stronger than g in the sense that f would be a strict implicant of g (in short, "Γ' does not subsume g").

Definition 1. *f is a strict implicant of g iff $f \models g$ but $g \not\models f$.*

Clearly, f is a strict implicant of g iff f is a strict subclause of g.

Definition 2. *Γ' subsumes g iff $\Gamma' \models f$ for some strict implicant f of g.*

At this point, some epistemological choices should be made about the intended role of Γ. We expect here that any formula in Γ can possibly be expelled in the transformation process leading to Γ'. Especially, when we need to differentiate between facts that are e.g. observations or additional circumstances that cannot be questioned (e.g. the observation that the light is on and that the switch is on), these facts must not be included within the set Γ that we are about to transform. To illustrate that, consider the following examples.

Example 1. Let $\Gamma = \{a, b, g = \neg a \vee \neg b \vee c\}$. Clearly, g represents the $a \wedge b \rightarrow c$ rule: Assume that it must prevail. Here, g is already in Γ. Note that c is a strict implicant of g. It must be the case that Γ' does not entail both a and b, otherwise $\Gamma' \models c$ and e.g. $\Gamma' \models a \rightarrow c$ which means that $a \wedge b \rightarrow c$ would be subsumed in Γ'.

Example 2. Let $\Gamma = \{\neg a, g = \neg a \vee \neg b \vee c\}$. Again, g represents the $a \wedge b \rightarrow c$ rule that is supposed to prevail. It must be the case that Γ' does not contain $\neg a$. Otherwise $\Gamma' \models f = \neg a \vee c$ with f representing $a \rightarrow c$ and subsuming g.

In the following, we assume that any formula in Γ can be expelled to make the g rule prevail. Let us consider a family of binary operators $\ominus$ that contract a consistent set Γ of formulas w.r.t. a formula h, giving rise to a set of formulas obeying the following constraints.

Definition 3. *A contraction operator $\ominus$ takes a consistent set of formulas Γ and a non-tautological formula h and delivers a set $\Gamma \ominus h$ of formulas s.t.*

- $\Gamma \ominus h \not\models h$
- $\Gamma \ominus h \subseteq Cn(\Gamma)$
- $\Gamma \ominus h = Cn(\Gamma \ominus h)$

The first condition entails that the formula that we are getting rid of from Γ should not belong to the logical consequences of the resulting set. The second condition entails that the resulting set is a subset of the logical consequences of Γ. In this way, a contraction operator cannot add any new information. The last condition entails that the resulting set is deductively closed.

Several contraction operators could be envisaged. Although no additional restriction need apply to $\ominus$, most acceptable operators will obey a principle of minimal change, altering Γ as little as possible.

When Γ is consistent, one natural candidate way to transform Γ into Γ' s.t. $\Gamma' \models g$ while Γ' does not subsume g would consist in contracting Γ w.r.t. all strict implicants f of g and then in inserting g so that g can be deduced. Actually, such an approach is wrong since inserting g might enable reasoning paths leading to f. Consider the following example.

Example 3. Let $\Gamma = \{c \rightarrow a \vee b\}$. Assume that we want to transform Γ so as to obtain Γ' s.t. $\Gamma' \models a \vee b \vee c$ and Γ' does not subsume $a \vee b \vee c$. Γ does not allow us to deduce any strict implicant of $a \vee b \vee c$. Many natural $\ominus$ operators deliver a set $\Gamma' = \Gamma$ when Γ is contracted w.r.t. all strict implicants of $a \vee b \vee c$. However, $\Gamma' \cup \{a \vee b \vee c\}$ clearly allows for $a \vee b$ to be deduced, which is a strict implicant of $a \vee b \vee c$.

4 A general solution

Accordingly, we need a family of $\oplus$ operators that turn a set Γ into a set Γ' entailing g although not subsuming g. However, these operators must be based on a more elaborate schema than the direct approach that we have just described.

Intuitively, the key observation leading to our solution is as follows. Let f be a strict implicant of g. We must expell f from Γ, as well as any way to derive f from Γ. If we expell $g \rightarrow f$ (which is equivalent to $\neg g \vee f$), then f is expelled. By doing so, we also prevent f to be derivable when g is actually added.

In the following, we often write $\Gamma \ominus \{h\}$ instead of $\Gamma \ominus h$ and assume that the set-theoretical union ($\cup$) operator on sets of formulas is always followed by deductive closure. The general solution should consider both cases where $\Gamma \cup \{g\}$ is (or is not) consistent.

CASE 1. $\Gamma \cup \{g\}$ is consistent.

Let us start with an $\oplus_f$ operator that deals with one *single* strict implicant of g.

Definition 4.1. *Let f be a strict implicant of g.*

$$\Gamma \oplus_f g =_{def} \Gamma \ominus \{g \to f\} \cup \{g\}$$

Theorem 1.

(1) *$\Gamma \oplus_f g$ is consistent.*
(2) $\Gamma \oplus_f g \models g$.
(3) $\Gamma \oplus_f g \not\models f$.

Proof. (1) From Definition 3, a property of $\ominus$ is that $\Gamma \ominus \{g \to f\} \not\models g \to f$. Then, $\Gamma \ominus \{g \to f\} \not\models \neg g \vee f$ hence $\Gamma \ominus \{g \to f\} \not\models \neg g$. Thus, $\Gamma \ominus \{g \to f\} \cup \{g\} \not\models \bot$.
(2) $\Gamma \oplus_f g \models g$ is trivial because $\Gamma \oplus_f g =_{def} \Gamma \ominus \{g \to f\} \cup \{g\}$.
(3) If $\Gamma \oplus_f g \models f$ then $\Gamma \ominus \{g \to f\} \cup \{g\} \models f$. Using the deduction theorem, $\Gamma \ominus \{g \to f\} \models g \to f$ which contradicts a property (Definition 3) of the $\ominus$ operator.

(2) and (3) ensure that g can be derived but is not subsumed by f in $\Gamma \oplus_f g$.

Let us now show how this approach can be extended to handle all strict implicants of g. Observe first that directly generalizing the above approach in the form $\Gamma \oplus g =_{def} \Gamma \ominus \{g \to \bigvee_i f_i\} \cup \{g\}$ where $\bigvee_i f_i$ denotes the disjonction of the strict implicants of g would fail since $\bigvee_i f_i$ is logically equivalent with g (remember that $\ominus$ applies only to non-tautological formulas).

As g is a clause, consisting of n different literals, it is sufficient to consider the n *prime implicants* of g, i.e. the n longest strict subclauses of g. Indeed, if Γ' does not allow any prime implicant of g to be derived, then no strict implicant of g can be deduced from Γ'. Therefore, we must resort to multiple contraction [7]. That is, $\ominus$ is now a binary operation over the powerset of clauses: Whenever Γ and Λ are sets of clauses, $\Gamma \ominus \Lambda$ yields a subset of (the clausal deductive closure of) Γ that entails no clause of Λ. Multiple contraction is characterized by the following four conditions:

(1) $\Gamma \ominus \Lambda \subseteq \Gamma$	(inclusion)
(2) if $\Lambda \cap Cn(\emptyset) = \emptyset$ then $\Lambda \cap \Gamma \ominus \Lambda = \emptyset$	(success)
(3) if $\Lambda \equiv_\Gamma \Theta$ then $\Gamma \ominus \Lambda = \Gamma \ominus \Theta$	(uniformity)
(4) if $\varphi \in \Gamma \ominus \Lambda$ then $\Gamma \ominus \Lambda \subseteq \Omega \subseteq \Gamma$ for some Ω s.t. $\Lambda \cap \Omega = \emptyset$ and $\Lambda \cap Cn(\Omega \cup \{\varphi\}) \neq \emptyset$	(relevance)

While the above is a characterization, a method to define an operation is as follows. Some notation is needed, where $\Gamma \bot \Lambda$ consists of all maximal subsets of Γ that do not entail any member of Λ. In symbols,

$$\Phi \in \Gamma \bot \Lambda \text{ iff } \begin{cases} \Phi \subseteq \Gamma \\ \Phi \cap \Lambda = \emptyset \\ \Gamma \cap Cn(\Omega) \neq \emptyset \text{ for all } \Omega \text{ s.t. } \Phi \subset \Omega \subseteq \Gamma \end{cases}$$

The next ingredient is a selection function μ (for any Γ and Λ, some subset of $\Gamma \bot \Lambda$ is elicited) that is any function such that the two conditions below must be satisfied:

$$- \emptyset \neq \mu(\Gamma \bot \Lambda) \subseteq \Gamma \bot \Lambda \text{ if } \Gamma \bot \Lambda \neq \emptyset$$
$$- \mu(\Gamma \bot \Lambda) = \Gamma \text{ if } \Gamma \bot \Lambda = \emptyset$$

Importantly, specifying a multiple contraction can be done by determining a selection function μ. Indeed, $\ominus$ is a multiple contraction operation iff there exists μ such that

$$\Gamma \ominus \Lambda = \bigcap \mu(\Gamma \perp \Lambda)$$

Then, $\oplus'$ can be defined as follows using multiple contraction:

Definition 4.2. *Let $f_1, \ldots, f_n$ be the prime implicants of g.*

$$\Gamma \oplus' g =_{def} \Gamma \ominus \{g \rightarrow f_i\}_{i=1..n} \cup \{g\}$$

Properties 4.2.

- $\Gamma \oplus' g$ is consistent.
- $\Gamma \oplus' g \models g$
- $\Gamma \oplus' g \not\models f$.

Thus g can be deduced but not subsumed by any of its prime implicants from $\Gamma \oplus g$.

It is easy to show that this definition enjoys the properties proven for the $\oplus_f$ operator (Theorem 1).

CASE 2. $\Gamma \cup \{g\}$ is inconsistent.

When $\Gamma \cup \{g\}$ is inconsistent, the $\oplus$ and $\oplus'$ operators do not apply, since $\ominus$ is undefined on inconsistent sets of formulas. In this specific case, we first require a preliminary revision step on Γ by g, by means of a semantical $*$ revision operator *à la* [4], which enforces consistency and the presence of g. Thereafter, a consistent set that already allows to derive g is obtained, and the $\oplus$ or $\oplus'$ operators can be applied to deliver a set that allows g to be derived but not to be subsumed.

By choosing a revision by g according to an AGM-like operator, we adopt a syntax-independent approach that allows for minimal changes and that preserves g.

Clearly, if the $\ominus$ operator was defined for sets of formulas that can be inconsistent, then the need to differentiate between CASE 1 and CASE 2 would disappear since the treatment proposed in CASE 1 would implicitly involve a revision step when needed, enforcing consistency and the ability to derive g.

Model-theory counterpart.

It is easy to provide an alternative -model-theoretic- $\oplus'$ operator in the general case.

Let $g = a_1 \vee \ldots \vee a_n$ where a_i are literals.

- $\mathcal{M}(\Gamma \oplus' g) \neq \emptyset$
- $\forall i \in [1 \ldots n]\ \exists\, m_i \in \mathcal{M}(\Gamma \oplus' g)$ s.t. $\{\neg a_1, \ldots, \neg a_n\} \setminus \{\neg a_i\} \subseteq m_i$
- $\mathcal{M}(\Gamma \cup \{g\}) \subseteq \mathcal{M}(\Gamma \oplus' g) \subseteq \mathcal{M}(\{g\})$

The first two items and the second part of the third one ensure the satisfaction of Properties 4.2 (the second item prevents any strict implicant of g from being derivable). The first part of the third item ensures that contraction operators used by $\oplus'$ deliver subsets of the deductive closure of Γ. Obviously enough, finer-grained model-theoretic characterizations will depend on the actual contraction operator that is selected.

5 Computational investigations

The approach proposed in this paper relies on AGM-related revision operators and on Boolean satisfiability checking, which both suffer from high worst-case complexity results since SAT-checking is NP-complete and AGM belief revision operators belong to the second level of the polynomial hierarchy [6]. However, recent progress in practical SAT solving allows some actual handling of many sets of clauses, although worst cases remain intractable unless P= NP.

The goal of this research was not to develop yet another experimental platform for Boolean revision-related issues. Instead, we have implemented a tool whose aim is simply to help a user understand all steps of the proposed process in this paper.

Roughly, when a user wants to insert a new clause g that should not be subsumed within an existing knowledge base Γ, g is checked to make sure that it is non-tautologi cal and consistent. Then, as an easy ad-hoc way to implement the general multiple contraction solution, the n longest strict subclauses f_i of g are then considered successively. For each f_i, the system checks the consistency of $\Gamma \cup \{\neg(g \rightarrow f_i)\}$. When inconsistency is detected, it is explained to the user how consistency could be restored to avoid g to be subsumed. Various AGM-like polices for restoring consistency are then explained to the user, focusing among other things on minimal changes policies

One salient feature of the platform is that it also provides the user with the interacting clauses leading to inconsistency, allowing him (her) to take his (her) decision based on the full knowledge of all aspects of the conflicts to be solved. Let us elaborate on this part of the platform and illustrate it by some experimental results. The platform allows detecting MUSes (Minimal Unsatisfiable Subsets of clauses), which are inconsistent sets of clauses that are such that dropping any one of their clauses restores consistency. MUSes represent the smallest "explanations" of inconsistency in terms of the number of involved clauses leading to a contradiction. Revision policies that eliminate formulas must drop at least one clause per MUS. Unsurprisingly, detecting MUSes and enumerating them is highly intractable in the worst case. Indeed, a set Γ of n clauses can exhibit $C_n^{n/2}$ MUSes in the worst case and checking whether a given formula belongs to the set of MUSes of another set of clauses or not is a Σ_2^p-hard problem [6].

However, when both the number of MUSes and their size remain small, recent progress in practical SAT-solving allows complete results to be obtained. In this respect, we claim that in many real-life situations, especially in decision-making systems, the size of the minimal chains of reasoning leading to a new inconsistency due to the insertion of a new clause often remains small with respect to the size of all the available knowledge and involves a limited number of clauses only. Moreover, in those systems, a new piece of information can often lead to a limited number of MUSes.

So, within some preset limited computational resources, for each f_i the platform aims at delivering the complete set of MUSes, which must be altered in order to ensure the consistency of $\Gamma \cup \{\neg(g \rightarrow f_i)\}$.

The platform is written in C++ and all experimentations have been run on a plain PC (Intel Core2Quad 2,66Ghz - RAM 4GB Linux Ubuntu 11.04 (2.6.38-8 generic)). Especially, the platform makes use of HYCAM [12] co-authored by one of the authors of the present paper, which is one of the most powerful algorithms to detect all MUSes of a set of Boolean clauses. It is an elaborate hybridization of some dedicated local search

and logically complete techniques. The efficiency of HYCAM on various instances, especially instances from the SAT competitions [3] have been published elsewhere [12] [13]. We have also experimented an alternative approach delivering a strict inconsistent cover of the instances, namely a sufficient number of MUSes that would restore consistency if dropped. Mainly, this approach iterates the following steps until consistency is proved: each time one MUS is found, it is taken away from the set of clauses and added to the cover under computation. Clearly, this second approach [11] is expected to prove more efficient in the general case since it does not necessarily compute all MUSes, since these latter ones can exhibit non-empty intersections.

First, we have focused on structured benchmarks from the last SAT international competition [3] in order to check the feasibility of the approach that would provide the user with the minimal sets of clauses leading to inconsistency and nominate the clauses that should be dropped in order for g to prevail. We begun with the initial step of the second case of the general approach from Section 4, which requires all MUSes or one strict inconsistent cover of $\Gamma \cup \{g\}$ to be found, under the circumstance that g is not necessarily interacting with those MUSes. Obviously, the approach only proved feasible for instances involving a limited number of MUSes of a limited size. Table 1 relates some typical results for those kinds of instances. The instance name is provided in the first column, followed by the numbers of clauses (#cla), variables (#var) and MUSes (#MUSes) of the instance. The last two main columns provide the experimental results for both the approach computing all MUSes and the approach providing just one strict cover of inconsistency. In each case, the time spent (#sec) to compute the results is provided, as well as the number of clauses in the discovered MUSes (#Cl-in-MUSes). Note that the negligible time spent for the usual pre-treatment that eliminates multiple-occurrences clauses is not taken into account in the results. For the covering approach, the number of detected MUSes (#MUSes) is also given. The time-out was set to 250 seconds.

				Finding all MUSes		Finding one MUSes-Covering		
instances	#cla	#var	#MUSes	#sec	#Cl-in-MUSes	#MUSes	#sec	#Cl-in-MUSes
barrel2	159	50	27	0.098	99	1	0.191	77
barrel3	942	275	67765	127	546	1	6.707	456
aim-100-1_6-no-4	160	100	1	0.084	48	1	0.153	48
aim-200-1_6-no-2	320	200	2	0.082	81	1	0.281	80
aim-200-2_0-no-4	400	200	2	0.084	43	1	0.432	42
dubois29	232	87	1	0.102	232	1	0.302	232
C168_FW_UT_851	6758	1909	102	1.798	30	1	47.12	8
C170_FR_RZ_32	4067	1659	32768	18.64	243	1	9.871	227
C202_FW_RZ_57	7434	1799	1	1.414	213	1	14.955	213
C220_FV_RZ_12	4017	1728	80272	5.057	56	1	21.607	11
C220_FV_SZ_65	4014	1728	103442	8.953	103	1	8.758	23

Table 1. A sample of experimental results for structured instances.

Actually, in the general case, many of benchmarks from the SAT competition proved experimentally intractable for our approach. Indeed, those difficult benchmarks have just been proposed to challenge current SAT solvers and many of the unsatisfiable instances involve very large MUSes. On the contrary, we assume that a new piece of information often contradicts only a limited part of the information present in a knowledge-based decision system, leading to a limited number of small minimal chains of reasoning leading to inconsistency. In this respect, benchmarks from the SAT solvers competition rarely match this intuition. Moreover, we also felt the need to investigate the influence of several keys parameters in the instances by making them vary, which is not simple or even possible using the above SAT benchmarks. Accordingly, we have designed our own instances generator that delivers unsatisfiable sets of clauses exhibiting a restricted number of MUSes of small sizes according to several parameters.

Table 2 provides some typical experimental results, showing the feasibility to provide the user with all MUSes encountered in the $\Gamma \cup \{\neg(g \rightarrow f_i)\}$ test, under the above assumptions. In the first column of Table 2, the various parameters of the generator are provided: namely, the number of MUSes (#MUSes), the size of each MUS, the size of g in terms of its number of literals, and the percentage of clauses in the instance not belonging to any MUS. The generator provides an instance Γ and a clause g according to these parameters: the number of clauses and variables of Γ are provided in the second main column. The last two main columns provide the experimental results for both the approach computing all MUSes and the approach just providing one strict cover of inconsistency. In each case, the time spent (#sec) to compute the results is provided, as well as the number of clauses in the discovered MUSes (#Cl-in-MUSes). For the covering approach, the number of detected MUSes (#MUSes) is also given. Again, the time-out was set to 250 seconds. The generator and all the discussed algorithms to enforce the predominance of g in Γ according to various policies are available at `http://www.cril.univ-artois.fr/~ramon/preempte`.

As expected, the approach finding out just one cover often allows more complex instances to be handled, as it still delivers results when the approach computing all MUSes reaches the time-out. For instance, the first time-out in the table is related to an instance made of 300 clauses involving 217 variables. g was made of three literals and it conflicts with the initial instance in 20 different minimal ways (MUSes), each of them involving 5 clauses. The covering approach detects one cover involving14 MUSes containing 48 different clauses within 4 seconds. As another example, we generated an instance involving 11 MUSes each of them of size 20, with an additional formula g made of 19 literals within an instance made of 210 clauses and 281 variables, 80% of those clause not being involved in the MUSes. The first approach delivered all MUSes involving all together 42 clauses within 19 seconds. The second approach delivered a cover made of 4 MUSes involving 41 clauses within 5 seconds. Let us stress that we believe that such an example involving a clause g of 19 literals is expected to represent a quite unusual situation as it involves a new very large decision rule to be added inside an already existing knowledge base. The last lines of the Table shows some results involving large knowledge bases (e.g. 4000 clauses involving 3801 variables), where the number of MUSes remains limited (5) as well as their size (5 clauses). The size of g being set to 3, both approaches delivered the final results in short computing times. Let

instances				Γ		Finding all MUSes		Finding one MUSes-Covering		
#MUSes	size(MUS)	size(g)	%not-in-MUSes	#cla	#var	#sec	#Cl-in-MUSes	#MUSes	#sec	#Cl-in-MUSes
1	5	3	80	15	24	<1	3	1	<1	3
2				30	37	<1	6	2	<1	6
3				45	46	<1	9	3	<1	9
4				60	57	<1	12	3	<1	10
5				75	66	<1	15	4	<1	13
6				90	77	<1	18	5	<1	16
7				105	86	<1	21	6	<1	19
8				120	97	3	24	6	<1	20
9				135	106	7	27	7	<1	23
10				150	117	36	30	7	2	24
20				300	217	*time out*		14	4	48
30				450	317	*time out*		21	11	72
40				600	417	*time out*		27	24	94
50				750	517	*time out*		34	45	118
5	10	3	80	200	191	<1	40	4	<1	33
	20			450	441	4	90		2	73
	30			700	691	25	140		3	113
	40			950	941	*time out*			6	153
	50			1200	1191	*time out*			8	193
	100			2450	2441	*time out*			43	393
	150			3700	3691	*time out*			112	593
	200			4950	4941	*time out*			231	793
5	10	3	80	200	191	<1	40	4	<1	33
		4		185	181	<1	37		<1	31
		5		170	171	<1	34		<1	29
		6		155	161	<1	31		<1	27
		7		140	151	<1	28		<1	25
		8		125	141	<1	25		<1	23
		9		110	131	<1	22		<1	21
	20	19		210	281	19	42		5	41
	30	29		310	431	*time out*			12	61
	40	39		410	581	*time out*			24	81
	50	49		510	731	*time out*			41	101
5	10	3	81	200	191	<1	40	4	<1	33
			83	200	191	<1			<1	
			85	240	229	<1			<1	
			87	280	267	<1			<1	
			89	360	343	<1			<1	
			91	440	419	<1			<1	
			93	560	533	<1			1	
			95	800	761	<1			2	
			97	1320	1255	<1			4	
			99	4000	3801	<1			22	

Table 2. A sample of experimental results for generated instances.

us observe that the covering approach proved less efficient here. We believe that this unexpected result is due to the lack of reuse of information by the covering approach that computes MUSes successively, which turns out to be a disadvantage when the number of MUSes is large and when, at the same time, each of them exhibits a small size within large knowledge bases.

6 Related works

Quite surprisingly, the issue in this paper has not received much attention in the literature. However, in some of our previous works [8] [9] we have addressed the problem of enforcing weaker rules within a specific nonmonotonic diagnosis-oriented framework, considering very specific cases of predominance only. Also, in [10] a limited-scope attempt to formalize the predominance of weaker formulas has been realized using a double belief revision schema. The approach proposed in this paper is totally different and covers the full clausal Boolean logic.

To some extent, the problem that is handled is this paper shares some similarities with the so-called "knowledge refinement" issue, which has been the object of many research efforts in the machine learning community (see e.g. the seminal works [16] [17]). In knowledge refinement, the goal is to refine a first-order rule in the presence of examples and/or counter-examples. The problem addressed in this paper is clearly different, since we are considering formulas and not mere facts as new pieces of information. Moreover, the difference is even more fundamental. Indeed, assume that we are trying to refine Γ in the light of g, where g is interpreted as an example or counter-example of Γ. Assume that g encodes the disjunction *The person I have seen was Peter, George or Andrew* and that Γ already entails *The person I have seen was either Peter or George*. The "example" g does not contradict Γ. Moreover, in logical terms, g does not bring any new information w.r.t. Γ since Γ already entails g. Accordingly, from a deductive point of view, g does not bring any new piece of information allowing Γ to be refined accordingly. Actually, we believe that the study presented in this paper could be a first step towards defining logic-based learning from examples frameworks that would allow logically weaker information to refine a stronger knowledge.

7 Conclusions and perspectives

This work can be understood as a way to extend the competence model of standard logic to a specific feature of reasoning and learning: how can logically weaker (but possibly more informative) knowledge be forced to prevail over existing logically stronger information. We envision several main directions to extend this work. First, this paper has focused on the clausal fragment, which allows a very simple treatment based on handling maximal subclauses to be obtained and allows us to benefit from SAT and MUS-based recent technologies. However, sticking to clauses does not allow us to address all possible predominance issues, as e.g. when we want a new incoming but less "precise" formula $a \rightarrow b$ to prevent the $a \rightarrow b \wedge c$ formula preexisting in Γ to be derivable anymore. Although the Definitions and Properties in the paper are not limited to the clausal fragment but apply for the full Boolean language, deriving a calculus

to handle predominance in a practical way for the full Boolean language remains an exercice to be done. Second, it would be natural to extend this framework to the finite first-order case, allowing for quantification. Also, we might want to extend the representation framework to nonmonotonic logics and defeasible rules. Indeed, it is well-known that rules often suffer from exceptions that can be expressed using consistency checks. A first account of the extension of this work to defeasible rules is provided in [5].

References

1. *SAT Live*. http://www.satlive.org/.
2. *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning*. LNCS, Springer, 2011.
3. *SAT Competition 2011*. http://www.satcompetition.org/2011/, 2011.
4. Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.
5. Philippe Besnard, Éric Grégoire, and Sébastien Ramon. Overriding subsuming rules. In *Proceedings of ECSQARU'11*, 2011.
6. Thomas Eiter and Georg Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, 1992.
7. André Fuhrmann and Sven Ove Hansson. A survey of multiple contractions. *Journal of Logic, Language and Information*, 3(1):39–76, 1994.
8. Éric Grégoire. Fusing cooperative technical-specification knowledge components. In *Proceedings of ICTAI'02*, pages 535–542, 2002.
9. Éric Grégoire. Fusing cooperative technical-specification knowledge components. *International Journal on Artificial Intelligence Tools*, 12(3):265–278, 2003.
10. Éric Grégoire. Knowledge refinement through revision. In *Proceedings of IEEE IRI'07*, pages 285–290, 2007.
11. Éric Grégoire, Bertrand Mazure, and Cédric Piette. Local-search extraction of muses. *Constraints*, 12(3):325–344, 2007.
12. Éric Grégoire, Bertrand Mazure, and Cédric Piette. Does this set of clauses overlap with at least one mus? In *CADE*, pages 100–115, 2009.
13. Éric Grégoire, Bertrand Mazure, and Cédric Piette. Using local search to find msses and muses. *European Journal of Operational Research*, 199(3):640–646, dec 2009.
14. Robert C. Moore. The role of logic in knowledge representation and commonsense reasoning. In *Proceedings of AAAI'82*, pages 428–433, 1982.
15. Allen Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.
16. Bradley L. Richards and Raymond J. Mooney. Automated refinement of first-order horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.
17. S. Wrobel. First order theory refinement. In *Advances in Inductive Logic Programming*, pages 14–33, IOS Press, Amsterdam, 1995.
18. Du Zhang and Éric Grégoire, editors. *Special issue on Managing and Reasoning in the Presence of Inconsistency*, International Journal of Semantic Computing, 2011 (to appear).