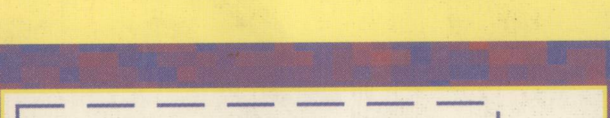
State-of-the-Art Survey

LNCS 3300

Leopoldo Bertossi
Anthony Hunter
Torsten Schaub (Eds.)

Inconsistency Tolerance



Springer

TG8
I37

Leopoldo Bertossi Anthony Hunter
Torsten Schaub (Eds.)

Inconsistency Tolerance

Springer

Volume Editors

Leopoldo Bertossi
Carleton University, School of Computer Science
Herzberg Building, 1125 Colonel By Drive, Ottawa, Canada K1S 5B6
E-mail: bertossi@scs.carleton.ca

Anthony Hunter
University College London, Department of Computer Science
Gower Street, London WC1E 6BT, UK
E-mail: a.hunter@cs.ucl.ac.uk

Torsten Schaub
Universität Potsdam, Institut für Informatik
August-Bebel-Str. 89,14482 Potsdam, Germany
E-mail: torsten@cs.uni-potsdam.de

Library of Congress Control Number: 2004117075

CR Subject Classification (1998): H.2, D.2, F.3, F.4

ISSN 0302-9743
ISBN 3-540-24260-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11373957 06/3142 5 4 3 2 1 0

Preface

The idea for this book arose after we had organized a meeting on inconsistency tolerance at Dagstuhl in Germany in the summer of 2003. We approached a number of eminent researchers in the field to contribute to the first book devoted to the subject. The net result is a collection of papers that provide an exciting coverage of some of the key aspects of the field.

All the chapters in the collection were anonymously reviewed, chapters by editors of the book being submitted for anonymous review by the other editors. Reviewing was undertaken by other authors involved in the project and by external reviewers. We are particularly grateful to the external reviewers as we believe they made a very significant contribution to all the chapters. The external reviewers included Ofer Arieli, Pablo Barcelo, Diego Calvanese, Sergio Greco, Jerome Lang, Domenico Lembo, Peter McBrien, Nic Wilson, and Peter Wood.

October 2004

Leo Bertossi
Anthony Hunter
Torsten Schaub

Table of Contents

Introduction to Inconsistency Tolerance

Leopoldo Bertossi[1], Anthony Hunter[2], and Torsten Schaub[3,*]

[1] School of Computer Science,
Carleton University,
1125 Colonel By Drive,
Ottawa, K1S 5B6, Canada
bertossi@scs.carleton.ca

[2] Department of Computer Science,
University College London
Gower Street, London WC1E 6BT, UK
a.hunter@cs.ucl.ac.uk

[3] Institut fur Informatik,
August-Bebel-Strasse 89,
D-14482 Potsdam, Germany
torsten@cs.uni-potsdam.de

Abstract. Inconsistency arises in many areas in advanced computing. Examples include: Merging information from heterogeneous sources; Negotiation in multi-agent systems; Understanding natural language dialogues; and Commonsense reasoning in robotics. Often inconsistency is unwanted, for example, in the specification for a plan, or in sensor fusion in robotics. But sometimes inconsistency is useful, e.g. when lawyers look for inconsistencies in an opposition case, or in a brainstorming session in research collaboration. Whether inconsistency is unwanted or useful, there is a need to develop tolerance to inconsistency in application technologies such as databases, knowledgebases, and software systems. To address this, inconsistency tolerance is being built on foundational technologies for identifying and analysing inconsistency in information, for representing and reasoning with inconsistent information, for resolving inconsistent information, and for merging inconsistent information. In this introductory chapter, we consider the need and role for inconsistency tolerance, and briefly review some of the foundational technologies for inconsistency tolerance.

1 The Need for Inconsistency Tolerance

Traditionally the consensus of opinion in the computer science community is that inconsistency is undesirable. Many believe that databases, knowledgebases, and software specifications, should be completely free of inconsistency, and try to eradicate inconsistency from them immediately by any means possible. Others

[*] Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada.

address inconsistency by isolating it, and perhaps resolving it locally. All seem to agree, however, that data of the form q and $\neg q$, for any proposition q cannot exist together, and that the conflict must be resolved somehow.

This view is too simplistic for developing robust software or intelligent systems, and furthermore, fails to use the benefits of inconsistent information in intelligent activities, or to acknowledge the fact that living with inconsistency seems to be unavoidable. Inconsistency in information is the norm in the real-world, and so should be formalized and used, rather than always rejected.

There are cases where q and $\neg q$ can be perfectly acceptable together and hence need not be resolved. Consider for example an income tax database where contradictory information on a taxpayer can be useful evidence in a fraud investigation. Maybe the taxpayer has completed one form that states the taxpayer has 6 children (hence the tax benefits for that) and completed another that states the taxpayer has 0 children. Here, this contradictory information needs to be kept and reasoned with. A similar example is in law courts where lawyers on opposing sides (for prosecution and defence) will seek contradictions in the opposition. Moreover, they will try to direct questions and to use evidence to engineer the construction of contradictions.

In other cases, q and $\neg q$ serve as a useful trigger for various logical actions. Inconsistency is useful in directing reasoning, and instigating the natural processes of argumentation, information seeking, multi-agent interaction, knowledge acquisition and refinement, adaptation, and learning.

In a sense, inconsistency can be seen as perfectly acceptable in a system, or even desirable in a system, as long as the system has appropriate mechanisms for acting on the inconsistencies arising [27]. Of course, there are inconsistencies that do need to be resolved. But, the decision to resolve, and the approach to resolution, need to be context-sensitive. There is also the question of when to resolve inconsistencies. Immediate resolution of inconsistencies can result in the loss of valuable information if an arbitrary choice is made on what to reject. Consider for example the requirements capture stage in software engineering. Here premature resolution can force an arbitary decision to be made without the choice being properly considered. This can therefore overly constrain the requirements capture process.

The call for robust, and intelligent, systems, has led to an increased interest in inconsistency tolerance in computer science. However, introducing inconsistency tolerance is a difficult and challenging aim. In the next section, we consider some of the problems, at the level of formal logic, arising from inconsistency. Then, in the subsequent section, we review a range of foundational technologies for use in developing inconsistency tolerance.

2 Problems Arising from Inconsistency

Classical mathematical logic is very appealing for knowledge representation and reasoning: The representation is rich and the reasoning powerful. Furthermore, classical reasoning is intuitive and natural. The appeal of classical logic however, extends beyond the naturalness of representation and reasoning. It has some very important and useful properties which mean that it is well-understood and well-behaved, and that it is amenable to automated reasoning.

Much of computer science is based on classical logic. Consider for example hardware logic, software specifications, SQL databases, and knowledgebase systems. Classical logic is therefore a natural starting point for considering inconsistency tolerance. Inconsistency is very much a logical concept, and so we should consider the effect of inconsistency on classical logic.

Unfortunately, inconsistency causes problems in reasoning with classical logic. In classical logic, anything can follow from an inconsistent set of assumptions. Let Δ be a set of assumptions, let $\vdash$ be the classical consequence relation, and let α be a formula, then $\Delta \vdash \alpha$ denotes that α is an inference from Δ using classical logic. A useful definition of inconsistency for a set of assumptions Δ is that if $\Delta \vdash \alpha$ and $\Delta \vdash \neg\alpha$ then Δ is inconsistent. A property of classical logic is that if Δ is inconsistent, then for any β in the language, $\Delta \vdash \beta$. This property results from the following proof rule, called *ex falso quodlibet*, being a valid proof rule of classical logic.

$$\frac{\alpha \qquad\qquad \neg\alpha}{\beta}$$

So inconsistency causes classical logic to collapse. No useful inferences follow from an inconsistent set of assumptions. It can be described as exploding, or trivialised, in the sense that all formulae of the language are consequences of inconsistent set of assumptions.

Since much of computer science is based on classical logic, the collapse of it in the face of inconsistency is a profound problem. We need to define the mechanisms for handling information in terms of a logic. So if classical logic is not appropriate for inconsistent information, we need to look elsewhere for a logic for inconsistency tolerance, or we need to consider mechanisms on top of classical logic to manage the information.

Even if we adopt a logic that does not collapse, i.e. ex falso quodlibet does not hold, we still need ways to handle the conflicting information. If we have a database that contains both α and $\neg\alpha$, we may need to answer the query "is α true?". An obvious strategy is that we only answer queries after we have cleaned the data by removing information to restore consistency. Another strategy is to take credulous approach to answering queries and so answer positively if the fact is in the database irrespective of the existence of its complement: In this case the answer would be "yes". A third strategy is to take a skeptical approach to answering queries and so answer positively if the fact is in the database and its complement is not: In this case the answer would be "no". A fourth strategy is a qualified credulous approach which refines the credulous inference with information about the existence of its complement.

The strategy of restoring consistency is not necessarily simple. For a set of formulae Δ, one option is to remove the union of the minimally inconsistent subsets to fix the inconsistency. Consider the set of beliefs.

$$\Delta = \{\alpha, \alpha \rightarrow \beta, \beta \rightarrow \gamma, \delta \rightarrow \neg\beta, \delta\}$$

There is only one minimally inconsistent subset of Δ:

$$\{\alpha, \alpha \rightarrow \beta, \delta \rightarrow \neg\beta, \delta\}.$$

To revise Δ, we can subtract the minimally inconsistent subset, and use the remainder as the revised knowledgebase. This is the same as taking the intersection of the maximally consistent subsets as the revised knowledgebase. So the revised knowledgebase is $\{\beta \rightarrow \gamma\}$. From this example, we see that the subtraction of the minimally inconsistent subset from the knowledgebase is quite drastic. An alternative is just to remove the smallest number of assumptions in order to restore consistency. Given Δ, we only need to remove one formula to restore consistency. There are four possible clauses we could choose for this:

$$\begin{gathered}\alpha \\ \alpha \rightarrow \beta \\ \delta \rightarrow \neg\beta \\ \delta\end{gathered}$$

So this gives us four choices for a revised set of assumptions. Each of these choices is a maximally consistent subset.

$$\begin{aligned}\Delta_1 &= \{\alpha, \beta \rightarrow \gamma, \delta \rightarrow \neg\beta, \delta\} \\ \Delta_2 &= \{\alpha, \alpha \rightarrow \beta, \beta \rightarrow \gamma, \delta\} \\ \Delta_3 &= \{\alpha, \alpha \rightarrow \beta, \beta \rightarrow \gamma, \delta \rightarrow \neg\beta\} \\ \Delta_4 &= \{\alpha \rightarrow \beta, \beta \rightarrow \gamma, \delta \rightarrow \neg\beta, \delta\}\end{aligned}$$

Clearly, such a revision is much more modest. But then we see we have a choice to make which may call for further knowledge and/or further strategies.

The conclusion we can draw from the discussions and examples in this section is that whilst classical logic is very useful in computer science, it needs to be adapted for use with inconsistent information, and that adapting it can involve some difficult issues. This has been the subject of much research, some of which we touch upon in the next section.

3 Foundational Technologies for Inconsistency Tolerance

Inconsistency tolerance is being built on foundational technologies for identifying and analysing inconsistency in information, for representing and reasoning with inconsistent information, for resolving inconsistent information, and for merging inconsistent information.

The central position is that the collapse of classical logic in cases of inconsistency should be circumvented. In other words, we need to suspend the principle of absurdity (*ex falso quodlibet*) for many kinds of reasoning. A number of useful proposals have been made in the field of paraconsistent logics.

In addition, we need strategies for analysing inconsistent information. This need has in part driven the approach of argumentation systems which compare pros and cons for potential conclusions from conflicting information. Also important are strategies for isolating inconsistency and for taking appropriate actions, including resolution actions. This calls for uncertainty reasoning and meta-level reasoning. Furthermore, the cognitive activities involved in reasoning with inconsistent information need to be directly related to the kind of inconsistency. So, in general, we see the need for inconsistency tolerance giving rise to a range of technologies for inconsistency management.

3.1 Consistency Checking

In order to manage inconsistency in knowledge, we need to undertake consistency checking. However, consistency checking is inherently intractable in the propositional case. To address this problem of the intractability, we can consider using (A) tractable subsets of classical logic (for example binary disjunctions of literals [30]), (B) heuristics to direct the search for a model (for example in semantic tableau [56], GSAT [67], and constraint satisfaction [22]), (C) some form of knowledge compilation (for example [53, 19]), and (D) formalization of approximate consistency checking based on notions described below, such as approximate entailment [49, 66], and partial and probable consistency.

Heuristic approaches, which have received a lot of attention in automated reasoning technologies and in addressing constraint satisfaction problems, can be either complete such as semantic tableau or Davis-Puttnam procedure [20] or incomplete such as in the GSAT system [68]. Whilst in general, using heuristics to direct search has the same worst-case computational properties as undirected search, it can offer better performance in practice for some classes of theories. Note, heuristic approaches do not tend to be oriented to offering any analysis of theories beyond a decision on consistency.

In approximate entailment, classical entailment is approximated by two sequences of entailment relations. The first is sound but not complete, and the second is complete but not sound. Both sequences converge to classical entailment. For a set of propositional formulae Δ, a formula α, and an approximate entailment relation $\models_i$, the decision of whether $\Delta \models_i \alpha$ holds or $\Delta \not\models_i \alpha$ holds can be computed in polynomial time.

Partial consistency takes a different approach to approximation. Furthermore, consistency checking for a set of formulae Δ can be prematurely terminated when the search space exceeds some threshold. When the checking of Δ is prematurely terminated, partial consistency is the degree to which Δ is consistent. This can be measured in a number of ways including the proportion of formulae from Δ that can be shown to form a consistent subset of Δ. Maximum generalized satisfiability [57] may be viewed as an example of this.

Yet another approach is probable consistency checking [40]. Determining the probability that a set of formulae is consistent on the basis of polynomial time classifications of those formulae. Classifications for the propositional case can be based on tests including counting the number of different propositional letters, counting the multiple occurrences of each propositional letter, and determining the degree of nesting for each logical symbol. The more a set of formulae is tested, the greater the confidence in the probability value for consistency, but this is at the cost of undertaking the tests.

Identifying approximate consistency for a set of formulae Δ is obviously not a guarantee that Δ is consistent. However, approximate consistency checking is useful because it helps to focus where problems possibly lie in Δ, and to prioritize resolution tasks. For example, if Δ and Γ are two parts of a larger knowledgebase that is thought to be inconsistent, and the probability of consistency is much greater for Δ than Γ, then Γ is more likely to be problematical and so should be examined more closely. Similarly, if Δ and Γ are two parts of a larger knowledgebase that is thought to be inconsistent, and a partial consistency identified for Δ is greater than for Γ, then Γ seems to contain more problematical data and so should be examined more closely by the user.

In databases, inconsistency is a notion relative to the satisfaction of a given set of integrity constraints (ICs), which are properties of the admissible database states. They impose semantic restrictions on the data in order to capture the correspondence of the data with the outside world that is being modelled by the database. We say that the database is inconsistent when the ICs, expressed as logical formulas, are not satisfied by the database, which can be seen as a first-order structure [64].

From this point of view, checking satisfaction of integrity constraints amounts to determining is a sentence is true in the given database. This can be easily done by posing and answering a query to/from the database. Taking into account that databases evolve as updates on it are executed, it becomes necessary to check every database state generated in this way. This process can be simplified using an inductive approach [54]: If the database was consistent before executing a certain update, then according to the kind of update and the kind of IC, it may be necessary to check only a formula that is much simpler that the original IC; or nothing at all if the update is irrelevant to the IC at hand [13]. Most approaches to consistency handling in database are directed to either detect potential inconsistencies, so that a problematic update is rejected before execution, or to accept the update even if an inconsistency is produced, but then detect or make a diagnosis of the data participating in the inconsistency, followed by an additional, remedial or compensating update that restores or enforces consistency [32, 16].

Clearly each approach to making consistency checking viable involves some form of compromise, and none is perfect for all applications. We therefore need a variety of approaches with clearly understood foundations and inter-relationships with other approaches. Furthermore, different techniques may give us different perspectives on inconsistencies in a given knowledgebase.

3.2 Paraconsistent Logics

Reasoning with inconsistency involves some compromise on the inferential machinery of classical logic. There is a range of proposals for logics (called paraconsistent logics) for reasoning with inconsistency. Each of the proposals has advantages and disadvantages. Selecting an appropriate paraconsistent logic for an application depends on the requirements of the application.

Types of paraconsistent logic that are proving to be of use for knowledge representation and reasoning in intelligent computing systems include: (1) Weakly-negative logics which use the full classical language, but a subset of the classical proof theory [21, 5]; (2) Four-valued logics which use a subset of the classical language and a subset of the classical proof theory, together with an intuitive four-valued semantics [6, 63, 4]; (3) Signed systems which involve renaming all literals in a theory and then restoring some of the original theory by progressively adding formal equivalences between the original literals and their renamings [10]; and (4) Quasi-classical logic which uses classical proof theory but restricts the notion of a natural deduction proof by prohibiting the application of elimination proof rules after the application of introduction proof rules [11, 35, 36].

These options behave in quite different ways with sets of assumptions. None can be regarded as perfect for handling inconsistent information in general. Rather, they provide a spectrum of approaches. However, in all the approaches the aim is to stay close to classical reasoning, since, as we have acknowledged, classical logic has many appealing features for knowledge representation and reasoning.

Paraconsistent logics are central to developing tolerance to inconsistency. Key research frontiers on this subject include: (1) developing a deeper understanding of the relationship of paraconsistency and substructural logics (for more information see Chapter 9 by John Slaney entitled "Relevant Logic and Paraconsistency"); (2) developing a deeper understanding of the computational complexity of paraconsistent logics (for more information see Chapter 6 by Sylvie Coste-Marquis and Pierre Marquis entitled "On the Complexity of Paraconsistent Inference Relations"); (3) developing automated reasoning technology for paraconsistent logics such via quantified Boolean formulae (for more information see Chapter 4 by Philippe Besnard, Torsten Schaub, Hans Tompits, and Stefan Woltran entitled "Representing Paraconsistent Reasoning via Quantified Boolean Formulae").

3.3 Argumentation Systems

Argumentation is an important cognitive activity that draws on conflicting knowledge for decision-making and problem solving. It normally involves identifying relevant assumptions and conclusions for a given problem being analysed. Furthermore, this often involves identifying conflicts, resulting in the need to look for pros and cons for particular conclusions. This may also involve chains of reasoning, where conclusions are used in the assumptions for deriving further conclusions. In other words, the problem may be decomposed recursively.

Coalition Systems. These are based on identifying sets of arguments that defend each other against counter-arguments by banding together for self-defence. The seminal proposal that can be described as using coalitions is by Dung [24]. This approach assumes a set of arguments, and a binary "attacks" relation between pairs of arguments. A hierarchy of arguments is then defined in terms of the relative attacks "for" and "against" each argument in each subset of the arguments. In this way, for example, the plausibility of an argument could be defended by another argument in its coalition (i.e. its subset).

Coherence Systems. One of the most obvious strategies for handling inconsistency in a knowledgebase is to reason with consistent subsets of the knowledgebase. This is closely related to the approach of removing information from the knowledgebase that is causing an inconsistency. In coherence systems, an argument is based on a consistent subset of a inconsistent set of formulae — the inconsistency arises from the conflicting views being represented. Further constraints, such as minimality or skeptical reasoning, can be imposed on the consistent subset for it to be an allowed argument. This range of further constraints gives us a variety of approaches to argumentation including [52, 14, 7, 8, 25, 2, 34, 12].

Defeasible Logics. There are a number of proposals for defeasible logics. The common feature for these logics is the incorporation of a defeasible implication into the language. Defeasible logics have their origins in philosophy and were originally developed for reasoning problems similar to those addressed by non-monotonic logics in artificial intelligence. In [59, 60], Pollock conceptualises the notions of reasons, prima facie reasons, defeaters, rebutting defeaters, and undercutting defeaters, in terms of formal logic. Arguments can then be defined as chains of reasons leading to a conclusion with consideration of potential defeaters at each step. Different types of argument occur depending on the nature of the reasons and defeaters. This has provided a starting point for a number of proposals for logic-based argumentation including abstract argument systems [71], conditional logic [55], and ordered logic [47].

There are many proposals for formalisms for logic-based argumentation. For general reviews of formalisms for argumentation see [31, 70, 61, 17]. Furthermore, some of these formalisms are being developed for applications in legal reasoning [62], in medical reasoning and risk assessment [26], and in agent-based systems [58]. A review of argumentation systems that relate proposals to potential application areas in knowledge engineering, decision-support, multi-agent negotiation, and software engineering, is given in [15].

3.4 Inconsistency Analysis

Given an inconsistent set of formulae Δ, we may need to know more about the nature of the inconsistency and the nature of information being offered by Δ. In some sense, we may desire inconsistency analysis based on notions that can be measured in Δ.

The seminal work on measuring inconsistency is by Shannon [69]. This work, based on probability theory, can be used in a logical setting when the worlds are the possible events. This work is also the basis of Lozinskii's work [51] for defining the quantity of information of a formula (or knowledgebase) in propositional logic. But this definition is not suitable when the knowledgebase is inconsistent. In this case, it has no classical model, so we have no "event" to count. To address this, models of maximal consistent subsets of the knowledgebase are considered.

Another related measure is the measure of contradiction. It is usual in classical logic to use a binary measure of contradiction: a knowledgebase is either consistent or inconsistent. This dichotomy is obvious when the only deductive tool is classical inference, since inconsistent knowledgebases are of no use. But, as we have identified earlier, there are now a number of paraconsistent logics developed to draw non-trivial conclusions from an inconsistent knowledgebase. So this dichotomy is not sufficient to describe the measure of contradiction of a knowledgebase, one needs more fine-grained measures.

Some interesting proposals have been made for this including: Consistency-based analyses that focus on the consistent and inconsistent subsets of a knowledgebase [39]; Information theoretic analyses that adapt Shannon's information measure [51, 72]; Probabilistic semantic analyses that consider maximal consistent probability distributions over a set of formulae [42, 43]; Epistemic actions analyses that measure the degree of information in a knowledgebase in terms of the number of actions required to identify the truth value of each atomic proposition and the degree of contradiction in a knowledgebase in terms of the number of actions needed to render the knowledgebase consistent [44]; and Model-theoretic analyses that are based on evaluating a knowledgebase in terms of three or four valued models that permit an "inconsistent" truth value [33, 37, 38].

This topic is the basis of Chapter 7 by Anthony Hunter and Sebastien Konieczny entitled "Approaches to Measuring Inconsistent Information".

3.5 Belief Revision

Given a knowledgebase Δ, and a revision α, belief revision theory is concerned with the properties that should hold for a rational notion of updating Δ with α. If $\Delta \cup \alpha$ is inconsistent, then belief revision theory assumes the requirement that the knowledge should be revised so that the result is consistent.

The AGM axioms, by Alchurron, Gardenfors and Makinson [1, 29], are postulates to delineate the behaviour of revision functions for belief sets (consider this as the set of all inferences obtained from a set of formulae). In the revision operation, as little of the belief set is changed as possible in order to include some new information. This requirement to change as little as possible precludes the change from a consistent set to an inconsistent set. In other words, some beliefs will be removed in order to maintain consistency.

The postulates appear as rational and intuitive properties that would be highly desirable. However, delivering efficient and effective systems that meet the postulates has proved to be challenging. There have been many developments of belief revision theory including iterated belief revision [18, 48], and relating belief revision to database updating [41]. These also offer intuitive abstract constraints for revision/updating. For a review of belief revision theory see [23].

There are some more concrete proposals for knowledgebase merging that adhere to belief revision postulates. In Konieczny and Pino Perez [45], there is a proposal for merging beliefs based on semantically characterizing interpretations which are "closest" to some sets of interpretations. But the approach does not