

# Introduction to Inconsistency Tolerance

Leopoldo Bertossi<sup>1</sup>, Anthony Hunter<sup>2</sup>, and Torsten Schaub<sup>3\*</sup>

<sup>1</sup> School of Computer Science  
Carleton University  
1125 Colonel By Drive  
Ottawa, K1S 5B6, Canada  
[bertossi@scs.carleton.ca](mailto:bertossi@scs.carleton.ca)

<sup>2</sup> Department of Computer Science  
University College London  
Gower Street, London WC1E 6BT, UK  
[a.hunter@cs.ucl.ac.uk](mailto:a.hunter@cs.ucl.ac.uk)

<sup>3</sup> Institut für Informatik  
August-Bebel-Strasse 89  
D-14482 Potsdam, Germany  
[torsten@cs.uni-potsdam.de](mailto: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 address inconsistency by isolating it, and perhaps resolving it locally. All seem

---

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

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 arbitrary 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  $\Delta$  be a set of assumptions, let  $\vdash$  be the classical consequence relation, and let  $\alpha$  be a formula, then  $\Delta \vdash \alpha$  denotes that  $\alpha$  is an inference from  $\Delta$  using classical logic. A useful definition of inconsistency for a set of assumptions  $\Delta$  is that if  $\Delta \vdash \alpha$  and  $\Delta \vdash \neg\alpha$  then  $\Delta$  is inconsistent. A property of classical logic is that if  $\Delta$  is inconsistent, then for any  $\beta$  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.

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

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  $\alpha$  and  $\neg\alpha$ , we may need to answer the query “is  $\alpha$  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  $\Delta$ , 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  $\Delta$ :

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

To revise  $\Delta$ , 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  $\Delta$ , we only need to remove one formula to restore consistency. There are four possible clauses we could choose for this:

$$\begin{array}{c} \alpha \\ \alpha \rightarrow \beta \\ \delta \rightarrow \neg\beta \\ \delta \end{array}$$

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  $\Delta$ , a formula  $\alpha$ , 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  $\Delta$  can be prematurely terminated when the search space exceeds some threshold. When the checking of  $\Delta$  is prematurely terminated, partial consistency is the degree to which  $\Delta$  is consistent. This can be measured in a number of ways including the proportion of formulae from  $\Delta$  that can be shown to form a consistent subset of  $\Delta$ . 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  $\Delta$  is obviously not a guarantee that  $\Delta$  is consistent. However, approximate consistency checking is useful because it helps to focus where problems possibly lie in  $\Delta$ , and to prioritize resolution tasks. For example, if  $\Delta$  and  $\Gamma$  are two parts of a larger knowledgebase that is thought to be inconsistent, and the probability of consistency is much greater for  $\Delta$  than  $\Gamma$ , then  $\Gamma$  is more likely to be problematical and so should be examined more closely. Similarly, if  $\Delta$  and  $\Gamma$  are two parts of a larger knowledgebase that is thought to be inconsistent, and a partial consistency identified for  $\Delta$  is greater than for  $\Gamma$ , then  $\Gamma$  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 if 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 than 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 an 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  $\Delta$ , we may need to know more about the nature of the inconsistency and the nature of information being offered by  $\Delta$ . In some sense, we may desire inconsistency analysis based on notions that can be measured in  $\Delta$ .

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  $\Delta$ , and a revision  $\alpha$ , belief revision theory is concerned with the properties that should hold for a rational notion of updating  $\Delta$  with  $\alpha$ . 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 exploit any meta-level information such as preferences. The approach has been generalized by considering merging with respect to integrity constraints [46].

Another approach that extends belief revision theory, called arbitration operators, is by Liberatore and Schaerf [50]. This is a form of merging restricted to merging only two knowledgebases and it forces the result to be the disjunction of the two original knowledgebases.

Proposals for belief revision that incorporate priorities include ordered theory presentations [65] and prioritized revision [28]. In ordered theory presentations, if a formula is less preferred than another which contradicts it, those aspects of it which are not contradicted are preserved. This is done by adopting an inferentially weaker formula to avoid the contradiction with the more preferred formula. This merging can be undertaken in an arbitrarily large partial ordering of formulae. In prioritized revision, a belief revision operator is defined in terms of selecting the model that satisfies the new belief and is nearest to the existing beliefs. The measure of nearness can be used in iterated belief revision where the more preferred items are used in later revisions.

Similar in spirit to belief revision is the recent work on consistent query answering in databases [3, 9]. The idea, as opposed to traditional approaches to inconsistency handling, is to live with an inconsistent database, but obtaining only consistent information (with respect to given integrity constraints) when queries are answered. That consistent information is the one that is invariant or persists under all possible minimal ways of restoring consistency of the database. There may be several alternative minimal *repairs* for a database, in consequence what is consistently true in a database instance is what is true in a collection of other instances that are the minimally repaired version of the original one. This approach shares many similarities with the problem of updating a database seen as a logical theory (or a model) by means of a set of sentences (the integrity constraints). In this case, the data is flexible, subject to repair, but the integrity constraints are hard, not to be given up. So, what is consistently true is what is true wrt to the revised database. A more precise comparison can be found in [3, 9].

## 4 Towards viable technologies

We are now at an exciting stage in the development of inconsistency tolerance. Rich foundations are being established, and a number of interesting and complementary application areas are being explored in decision-support, multi-agent systems, database systems, and software engineering.

Key frontiers in developing viable applications technologies include: Integrating data from heterogeneous databases (for more information see Chapter 3 by Leo Bertossi and Loreto Bravo entitled “Consistent Query Answers in Virtual Data Integration Systems”); Computational complexity issues in integrity maintenance (for more information see Chapter 5 by Jan Chomicki and Jerzy Marcinkowski entitled “On the Computational Complexity of Minimal-Change

Integrity Maintenance in Relational Databases”; Representing and reasoning with spatial data (for more information see Chapter 8 by Andrea Rodriguez entitled ”Inconsistency Issues in Spatial Databases”; and Computational complexity issues in handling XML specifications (for more information see Chapter 2 by Marcelo Arenas, Leonid Libkin and Wenfei Fan entitled ”Consistency of XML specifications”).

## 5 Conclusions

In this introduction, we have highlighted the need for inconsistency tolerance in order to create more robust and more intelligent computing systems. Inconsistency tolerance is being built on foundational technologies of identifying and analysing inconsistency in information, for representing and reasoning with inconsistent information, for resolving inconsistent information, and for merging inconsistent information. Inconsistency tolerance is now being developed for a range of applications in database, knowledgebase and software systems.

## References

1. C Alchourron, P Gardenfors, and D Makinson. On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.
2. L Amgoud and C Cayrol. On the acceptability of arguments in preference-based argumentation. In G Cooper and S Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1998.
3. M Arenas, L Bertossi, and J Chomicki. Consistent query answers in inconsistent databases. In *Proc. ACM Symposium on Principles of Database Systems (PODS 99)*, pages 68–79, 1999.
4. O Arieli and A. Avron. The value of the four values. *Artificial Intelligence*, 102:97–141, 1998.
5. D Batens. Paraconsistent extensional propositional logics. *Logique et Analyse*, 90–91:195–234, 1980.
6. N Belnap. A useful four-valued logic. In G Epstein, editor, *Modern Uses of Multiple-valued Logic*, pages 8–37. Reidel, 1977.
7. S Benferhat, D Dubois, and H Prade. Argumentative inference in uncertain and inconsistent knowledge bases. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 1449–1445. Morgan Kaufmann, 1993.
8. S Benferhat, D Dubois, and H Prade. A logical approach to reasoning under inconsistency in stratified knowledge bases. In *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, volume 956 of *Lecture Notes in Computer Science*, pages 36–43. Springer, 1995.
9. L Bertossi and J Chomicki. Query answering in inconsistent databases. In G Saake J Chomicki and R van der Meyden, editors, *Logics for Emerging Applications of Databases*. Springer, 2003.
10. Philippe Besnard and Torsten Schaub. Signed systems for paraconsistent reasoning. *Journal of Automated Reasoning*, 20:191–213, 1998.

11. Ph Besnard and A Hunter. Quasi-classical logic: Non-trivializable classical reasoning from inconsistent information. In C Froidevaux and J Kohlas, editors, *Symbolic and Quantitative Approaches to Uncertainty*, volume 946 of *Lecture Notes in Computer Science*, pages 44–51, 1995.
12. Ph Besnard and A Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128:203–235, 2001.
13. J Blakeley, N Coburn, and P Larson. Updating derived relations: detecting irrelevant and autonomously computable updates. *ACM Transactions on Database Systems*, 14(3):369–400, 1989.
14. G Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *Proceedings of the Eleventh International Conference on Artificial Intelligence*, pages 1043–1048, 1989.
15. D Carbogim, D Robertson, and J Lee. Argument-based applications to knowledge engineering. *Knowledge Engineering Review*, 15:119–149, 2000.
16. S Ceri, P Fraternali, S Paraboschi, and L Tanca. Automatic generation of production rules for integrity maintenance. *ACM Transactions on Database Systems*, 19(3):367–422, 1994.
17. C Chesnevar, A Maguitman, and R Loui. Logical models of argument. *ACM Computing Surveys*, 32:337–383, 2001.
18. A Darwiche and J Pearl. On the logic of iterated belief revision. *Artificial Intelligence*, 89:1–29, 1997.
19. A Darwiche. Compiling knowledge into decomposable negation normal form. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 284–289, 1999.
20. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
21. N C da Costa. On the theory of inconsistent formal systems. *Notre Dame Journal of Formal Logic*, 15:497–510, 1974.
22. R Dechter and J Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.
23. D Dubois and H Prade, editors. *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 3. Kluwer, 1998.
24. P. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.
25. M Elvang-Goransson and A Hunter. Argumentative logics: Reasoning from classically inconsistent information. *Data and Knowledge Engineering*, 16:125–145, 1995.
26. J Fox and S Das. *Safe and Sound: Artificial Intelligence in Hazardous Applications*. MIT Press, 2000.
27. D Gabbay and A Hunter. Making inconsistency respectable 1: A logical framework for inconsistency in reasoning. In *Fundamentals of Artificial Intelligence*, volume 535 of *Lecture Notes in Computer Science*, pages 19–32. Springer, 1991.
28. D Gabbay and O Rodrigues. A methodology for iterated theory change. In *Practical Reasoning*, volume 1085 of *Lecture Notes in Computer Science*. Springer, 1996.
29. P Gardenfors. *Knowledge in Flux*. MIT Press, 1988.
30. M Garey and D Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
31. J Gebhardt and R Kruse. Background and perspectives of possibilistic graphical models. In A Hunter and S Parsons, editors, *Applications of Uncertainty Formalisms*, Lecture Notes in Computer Science. Springer, 1998.

32. M Gertz and W Lipeck. An extensible framework for repairing constraint violations. In S Jajodia et al., editor, *Integrity and Internal Control in Information Systems, IFIP TC11 Working Group 11.5, First Working Conference on Integrity and Internal Control in Information Systems: Increasing the confidence in Information Systems, Zurich, Switzerland, December 4-5, 1997*, pages 89–111. Chapman Hall, 1997.
33. J Grant. Classifications for inconsistent theories. *Notre Dame Journal of Formal Logic*, 19:435–444, 1978.
34. R Haenni, J Kohlas, and N Lehmann. Probabilistic argumentation systems. In D Gabbay and Ph Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems, Volume 5*, pages 221–288. Kluwer, 2000.
35. A Hunter. Reasoning with contradictory information using quasi-classical logic. *Journal of Logic and Computation*, 10:677–703, 2000.
36. A Hunter. A semantic tableau version of first-order quasi-classical logic. In *Quantitative and Qualitative Approaches to Reasoning with Uncertainty*, LNCS. Springer, 2001. 544–556.
37. A Hunter. Measuring inconsistency in knowledge via quasi-classical models. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'2002)*, pages 68–73. MIT Press, 2002. ISBN 0-262-51129-0.
38. A Hunter. Evaluating the significance of inconsistency. In *Proceedings of the International Joint Conference on AI (IJCAI'03)*, pages 468–473, 2003.
39. A Hunter. Logical comparison of inconsistent perspectives using scoring functions. *Knowledge and Information Systems Journal*, 2004. (in press).
40. A Hunter. Probable consistency checking for sets of propositional clauses. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, volume 2711 of Lecture Notes in Computer Science. Springer, 2003. pages 464 - 476.
41. H Katsuno and A Mendelzon. On the difference between updating a knowledgebase and revising it. *Belief Revision*, pages 183–203, 1992.
42. K Knight. Measuring inconsistency. *Journal of Philosophical Logic*, 31:77–98, 2001.
43. K Knight. Two information measures for inconsistent sets. *Journal of Logic, Language and Information*, 12:227–248, 2003.
44. S Konieczny, J Lang, and P Marquis. Quantifying information and contradiction in propositional logic through epistemic actions. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003. in press.
45. S Konieczny and R Pino Perez. On the logic of merging. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR98)*, pages 488–498. Morgan Kaufmann, 1998.
46. S Konieczny and R Pino Perez. Merging with integrity constraints. In Anthony Hunter and Simon Parsons, editors, *Qualitative and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU'99)*, volume 1638 of *Lecture Notes in Computer Science*. Springer, 1999.
47. E Laenens and D Vermeir. A fixpoint semantics for ordered logic. *Journal of Logic and Computation*, 1:159–185, 1990.
48. D Lehmann. Belief revision, revised. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1534–1540, 1995.
49. H Levesque. A logic of implicit and explicit belief. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'84)*, pages 198–202, 1984.
50. P Liberatore and M Schaerf. Arbitration (or how to merge knowledgebases). *IEEE Transactions on Knowledge and Data Engineering*, 10:76–90, 1998.
51. E Lozinskii. Information and evidence in logic systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:163–193, 1994.

52. R Manor and N Rescher. On inferences from inconsistent information. *Theory and Decision*, 1:179–219, 1970.
53. P Marquis. Knowledge compilation using prime implicants. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 837–843, 1995.
54. J-M. Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18:227–253, 1982.
55. D Nute. Defeasible reasoning and decision support systems. *Decision Support Systems*, 4:97–110, 1988.
56. F Oppacher and E Suen. HARP: A tableau-based theorem prover. *Journal of Automated Reasoning*, 4:69–100, 1988.
57. C Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
58. S Parsons, C Sierra, and N Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8:261–292, 1998.
59. J Pollock. Defeasible reasoning. *Cognitive Science*, 11:481–518, 1987.
60. J Pollock. How to reason defeasibly. *Artificial Intelligence*, 57:1–42, 1992.
61. H Prakken and G Vreeswijk. Logical systems for defeasible argumentation. In D Gabbay, editor, *Handbook of Philosophical Logic*. Kluwer, 2000.
62. H Prakken. *Logical Tools for Modelling Legal Argument*. Kluwer, 1997.
63. G Priest. Reasoning about truth. *Artificial Intelligence*, 39:231–244, 1989.
64. R Reiter. Towards a logical reconstruction of relational database theory. In M. Brodie, J. Mylopoulos, and J. Schmidt, editors, *On Conceptual Modeling*, pages 191–233. Springer-Verlag, 1984.
65. M Ryan. Representing defaults as sentences with reduced priority. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*. Morgan Kaufmann, 1992.
66. M Schaerf and M Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74:249–310, 1995.
67. B Selman, H Levesque, and D Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, 1992.
68. B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In P. Rosenbloom and P. Szolovits, editors, *Proceedings of AAAI'92*, pages 440–446. AAAI Press, 1992.
69. C Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.
70. D Vermeir, E Laenens, and P Geerts. Defeasible logics. In *Handbook of Defeasible Reasoning and Uncertainty Management*, volume 2. Kluwer, 1998.
71. G Vreeswijk. Abstract argumentation systems. *Artificial Intelligence*, 90:225–279, 1997.
72. P Wong and Ph Besnard. Paraconsistent reasoning as an analytic tool. *Journal of the Interest Group in Propositional Logic*, 9:233–246, 2001.