Probable Consistency Checking
for Classical Propositional Logic

Anthony Hunter
Department of Computer Science
University College London
Gower Street
London WC1E 6BT, UK

Email:a.hunter@cs.ucl.ac.uk
Tel:+44 20 7679 7295
Fax:+44 20 7387 1397

October 18, 2002

Abstract

Inconsistencies inevitably arise in knowledge during practical reasoning. In a logic-based approach this gives rise to the need for consistency checking. Unfortunately, this can be difficult. In classical propositional logic, this is intractable. However, there is a useful alternative to the notion of consistency called probable consistency. This offers a weakening of classical consistency checking where polynomial time tests are done on a formula to determine the probability that the formula is consistent. In this paper, we present a framework for probable consistency checking, and analyse some classes of polynomial time tests. We then consider how a probable consistency checking system can be used in applicatons such as monitoring negotiations, and monitoring newsfeeds, where there is a heavy demand for consistency checking.

Index terms: Consistency checking, Logic-based negotiation, Logic-based information integration

1 Introduction

Classical 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.

However, in reasoning about the real-world, it is common to have “too much” information about some situation. In other words, it is common for there to be classically inconsistent information in a practical reasoning situation [GH91]. Unfortunately, by the definition of classical logic, the reasoning is trivialized because any inference follows from inconsistent information, by the *ex falso quodlibet* proof rule.

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 [GJ79]), (B) heuristics to direct the search for a model[1] (for example in semantic tableau [OS88], GSAT [SLM92], and constraint satisfaction [DP87]), (C) some form of knowledge compilation (for example [Mar95, Dar99]), and (D) formalization of approximate consistency checking based on notions described below, such as approximate entailment, and partial and probable consistency.

Approximate entailment Proposed in [Lev84], and developed in [SC95], 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 Consistency checking does not necessarily involve an exponential search space. 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 [Pap94] may be viewed as an example of this.

Probable consistency 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/inconsistency, 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.

The notions of probable consistency and partial consistency provide complementary means for reasoning with knowledge. For a set of formulae Δ, with partial consistency, we may identify subsets of Δ that are consistent, but have no certainty on whether Δ is consistent, whereas with probable consistency, we can obtain a view on the whole of Δ, but cannot guarantee to find consistent subsets even if they exist.

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. The need to analyse inconsistencies is a key motivation for probable consistency checking and in this role can complement techniques such as those based on measuring coherence [Hun02b]. So in addition

[1] Heuristic approaches can be either complete such as semantic tableau or incomplete such as in the GSAT system. 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.

to a decision on whether any of the formulae in a knowledgebase is approximately inconsistent, we seek further information characterizing the nature of any of the inconsistencies arising.

In this paper, we explore probable consistency checking in more detail. We focus the exposition on the probable consistency of individual formulae rather than sets of formulae. In a follow-on paper, we extend our framework to sets of formulae. In the next section, we provide some basic definitions for syntax, then in Section 3, we present the framework for probable consistency checking. In Section 4, we illustrate the framework by analysing some restricted languages for probable consistency checking. Then in Section 5, we consider the viability of probable consistency checking. Finally, in Section 6, we consider application areas for probable consistency checking including: (1) Highlighting potentially problematical information such as in monitoring requirements during negotiations and monitoring data feeds such as newsfeeds; and (2) Using probable consistency as the basis of an analytical framework complementing empirical frameworks based on satisfiability testing.

2 Basic definitions for syntax

Let $\mathbb{N}$ denote the set of natural numbers and let $\mathbb{N}^+$ denote the positive integers. Also let $\wp(X)$ denote the powerset for any set X.

Definition 2.1 *Let $\mathcal{L}_{atoms}$ be a set of* **atoms** *and let $\mathcal{L}$ be the set of* **classical propositional formulae** *formed from $\mathcal{L}_{atoms}$, and the $\wedge, \vee$ and $\neg$ connectives. For each atom $\alpha \in \mathcal{L}_{atoms}$, α is a* **positive literal** *and $\neg\alpha$ is a* **negative literal**. *Let $\mathcal{L}_{literals}$ be the set of literals.*

Example 2.1 *From the propositional atoms α, β and γ, members of $\mathcal{L}$ include $\alpha, \beta \wedge \gamma$, $\neg\alpha \wedge \alpha$ and $(\alpha \wedge \beta) \vee \neg\neg\gamma$.*

To simplify our exposition we do not consider the implication connective. However, we can easily generalize the framework presented in this paper to include it.

Definition 2.2 *For $\alpha_1 \vee .. \vee \alpha_n \in \mathcal{L}$, $\alpha_1 \vee .. \vee \alpha_n$ is a* **clause** *iff each of $\alpha_1, .., \alpha_n$ is a literal. For $\beta_1 \wedge .. \wedge \beta_n \in \mathcal{L}$, $\beta_1 \wedge .. \wedge \beta_n$ is in* **conjunctive normal form** *(CNF) iff each of $\beta_1, .., \beta_n$ is a clause. For clauses, $\alpha \vee \beta$, $\neg\beta \vee \gamma$, $\alpha \vee \gamma \in \mathcal{L}$, $\alpha \vee \gamma$ is a* **resolvent** *of $\alpha \vee \beta$ and $\neg\beta \vee \gamma$.*

Definition 2.3 *Let $\vdash$ denote the* **classical consequence relation**. *Let $\perp$ denote any inconsistent formula in $\mathcal{L}$. If $\{\phi\} \vdash \perp$ holds, then ϕ is an inconsistent formula. If $\{\phi\} \not\vdash \perp$ holds, then ϕ is a consistent formula. Let $\mathcal{L}_{inconsistent}$ be the subset of $\mathcal{L}$ containing all the inconsistent formulae in $\mathcal{L}$ and let $\mathcal{L}_{consistent}$ be the subset of $\mathcal{L}$ containing all the consistent formulae in $\mathcal{L}$.*

Definition 2.4 *Let the* **atoms function**, *denoted $atoms$, be a function from $\mathcal{L}$ into $\mathcal{L}_{atoms}$ such that $atoms(\alpha)$ gives the set of atoms used in α.*

Example 2.2 *For $\alpha, \beta \in \mathcal{L}_{atoms}$, $atoms(\alpha \wedge \beta) = \{\alpha, \beta\}$, $atoms(\alpha \wedge \alpha) = \{\alpha\}$, and $atoms((\neg\alpha \wedge \gamma) \vee \alpha) = \{\alpha, \gamma\}$.*

Definition 2.5 *Let $\mathcal{A}$ be a finite subset of $\mathcal{L}_{atoms}$. Let $\mathcal{L}^{\mathcal{A}} = \{\alpha \in \mathcal{L} \mid atoms(\alpha) \subseteq \mathcal{A}\}$.*

So, $\mathcal{L}^{\mathcal{A}}_{atoms} = \mathcal{A}$, and $\mathcal{L}^{\mathcal{A}}_{literals} = \mathcal{A} \cup \{\neg\alpha \mid \alpha \in \mathcal{A}\}$.

3 Probable consistency checking

In this section, we first consider the subsidiary issues of restricted languages and test functions, and then use these to define the notion of probable consistency.

3.1 Using restricted languages

Given a formula $\phi \in \mathcal{L}$, we seek a probability that ϕ is consistent. To do this, we work with analysed subsets of $\mathcal{L}$. Essentially, if ϕ is a member of a particular subset $\mathcal{L}_r$ of $\mathcal{L}$, and we know the proportion of $\mathcal{L}_r$ that are inconsistent is p, then we use p as the probability that ϕ is inconsistent. For example, consider the set $\mathcal{L}_1$.

$$\mathcal{L}_1 = \{\alpha \wedge \neg\alpha, \beta \wedge \neg\beta, \gamma \wedge \neg\gamma, \delta \wedge \neg\delta, \varepsilon \wedge \neg\varepsilon\}$$

If ϕ is a member of $\mathcal{L}_1$, then the probability of inconsistency is 1. Whereas, if ϕ is a member of $\mathcal{L}_2$, then the probability of inconsistency is 0.

$$\mathcal{L}_2 = \{\alpha \vee \neg\alpha, \beta \vee \neg\beta, \gamma \vee \neg\gamma, \delta \vee \neg\delta, \varepsilon \vee \neg\varepsilon\}$$

A restricted language $\mathcal{L}_r$ is a subset of the classical language $\mathcal{L}$ according to some syntactic criterion. For any $\mathcal{A} \subseteq \mathcal{L}_{atoms}$, and for any $\mathcal{L}_r$, let $\mathcal{L}_r^{\mathcal{A}} = \mathcal{L}^{\mathcal{A}} \cap \mathcal{L}_r$. There are many syntactic criteria for selecting a set $\mathcal{L}_r$. These include delineating one or more of the following: (1) The maximum number of different propositional letters; (2) The maximum number of occurrences of any propositional letter; (3) The length of the formula; (4) The maximum number of occurrences of each logical symbol; or (5) The maximum degree of nesting of each logical symbol. Using this approach, we can identify finite subsets of $\mathcal{L}$. Finiteness renders the approach more practical. We assume every formula in the restricted language $\mathcal{L}_{inconsistent}$ is an inconsistent formula and every formula in the restricted language $\mathcal{L}_{consistent}$ is a consistent formula.

Example 3.1 *Let $\mathcal{L}_{example1}$ be restricted to formulae with only the atom α, with multiple occurrences allowed, and maximum occurrence of one $\neg$ symbol and a maximum occurrence of one $\vee$ symbol.*

$$\begin{array}{ccc} \alpha & \alpha \vee \alpha & \neg(\alpha \vee \alpha) \\ \neg\alpha & \neg\alpha \vee \alpha & \alpha \vee \neg\alpha \end{array}$$

While there are logically equivalent formulae in $\mathcal{L}_{example1}$, this restricted language contains 6 syntactically different formulae. Note, none of the formulae in $\mathcal{L}_{example1}$ is inconsistent.

Example 3.2 *Let $\mathcal{L}_{example2}$ be restricted to formulae with only the atom α, with multiple occurrences allowed, and maximum occurrence of one $\neg$ symbol and a maximum occurrence of one $\wedge$ symbol. Note, 2 of the 6 formulae in $\mathcal{L}_{example2}$ are inconsistent.*

$$\begin{array}{ccc} \alpha & \alpha \wedge \alpha & \neg(\alpha \wedge \alpha) \\ \neg\alpha & \neg\alpha \wedge \alpha & \alpha \wedge \neg\alpha \end{array}$$

Example 3.3 *Let $\mathcal{L}_{example3}$ be restricted to formulae formed from the atoms $\{\alpha, \beta\}$, with multiple occurrences allowed, and maximum occurrence of one $\neg$ symbol and a maximum occurrence of one $\wedge$ symbol. Note, 4 of the 20 formulae in $\mathcal{L}_{example3}$ are inconsistent.*

$$\begin{array}{cccc}
\alpha & \beta & \neg\alpha & \neg\beta \\
\alpha \wedge \alpha & \alpha \wedge \beta & \beta \wedge \alpha & \beta \wedge \beta \\
\neg(\alpha \wedge \alpha) & \neg(\alpha \wedge \beta) & \neg(\beta \wedge \alpha) & \neg(\beta \wedge \beta) \\
\neg\alpha \wedge \alpha & \alpha \wedge \neg\alpha & \beta \wedge \neg\beta & \neg\beta \wedge \beta \\
\alpha \wedge \neg\beta & \neg\alpha \wedge \beta & \beta \wedge \neg\alpha & \neg\beta \wedge \alpha
\end{array}$$

Example 3.4 *Let $\mathcal{L}_{example4}$ be restricted to formulae formed from the atoms $\{\alpha, \beta\}$ and the logical symbols $\{\neg, \vee, \wedge\}$, where each formulae $\phi \in \mathcal{L}_{example4}$ has 4 symbols (propositional letters and logical symbols). Note, 4 of the 16 formulae in $\mathcal{L}_{example4}$ are inconsistent.*

$$\begin{array}{cccc}
\neg\alpha \wedge \alpha & \alpha \wedge \neg\alpha & \neg\alpha \vee \alpha & \alpha \vee \neg\alpha \\
\neg\alpha \wedge \beta & \alpha \wedge \neg\beta & \neg\alpha \vee \beta & \alpha \vee \neg\beta \\
\neg\beta \wedge \alpha & \beta \wedge \neg\alpha & \neg\beta \vee \alpha & \beta \vee \neg\alpha \\
\neg\beta \wedge \beta & \beta \wedge \neg\beta & \neg\beta \vee \beta & \beta \vee \neg\beta
\end{array}$$

Even for a small number of atoms, and a small degree of nesting, there are many formulae that can be formed. As a result, the number of different restricted languages that are possible is enormous. However, we do have systematic means for generating restricted languages, and in particular for determining the cardinality of these sets, and for determining the proportion of inconsistent formulae. We return to this issue in Section 3.4.

3.2 Test functions

In order to categorize formulae, we use functions, called test functions. A test function, or combination of test functions, can be used to define the membership function for a restricted language. We present some polynomial time tests below and in Table 1.

Definition 3.1 *The test function $T_{atoms}(\alpha)$ gives the number of different propositional letters used in the formula α. So $T_{atoms}(\alpha) = |atoms(\alpha)|$.*

Definition 3.2 *The test $T_{poslit}(\alpha)$ gives the number of occurrences of positive literals in α, and the test $T_{neglit}(\alpha)$ gives the number of occurrences of negative literals in α.*

Clearly the test functions in Definitions 3.1 and 3.2, and Table 1, are polynomial time functions. Polynomial time test functions can also be defined to determine whether a formula is a literal, a conjunction of literals, a clause, a conjunction of clauses, and so on. We also can use a battery of such test functions as polynomial test functions to determine whether a formula is a member of a particular restricted language.

Example 3.5 *Consider ϕ as $\delta \vee (\beta \wedge \neg(\gamma \vee \neg\beta))$.*

$$T_{atoms}(\phi) = 3 \quad T_{count\neg}(\phi) = 2 \quad T_{count\wedge}(\phi) = 1 \quad T_{count\vee}(\phi) = 2$$

Example 3.6 *Consider ψ as $\alpha \vee \neg\alpha$.*

$$\begin{array}{c}
T_{atoms}(\psi) = 1 \quad T_{count\neg}(\psi) = 1 \quad T_{count\wedge}(\psi) = 0 \quad T_{count\vee}(\psi) = 1 \\
T_{poslit}(\psi) = 1 \quad T_{neglit}(\psi) = 1
\end{array}$$

These tests therefore show that ψ is in the language $\mathcal{L}_{example1}$ described in Example 3.1

Test function	Definition
$T_{count\neg}(\alpha)$	Number of occurrences of $\neg$ in α
$T_{count\wedge}(\alpha)$	Number of occurrences of $\wedge$ in α
$T_{count\vee}(\alpha)$	Number of occurrences of $\vee$ in α

Table 1: Definitions for some test functions.

Language	Definition
$\mathcal{L}_{count\neg=p}$	$\{\alpha \in \mathcal{L} \mid T_{count\neg}(\alpha) = p\}$
$\mathcal{L}_{count\wedge=p}$	$\{\alpha \in \mathcal{L} \mid T_{count\wedge}(\alpha) = p\}$
$\mathcal{L}_{count\vee=p}$	$\{\alpha \in \mathcal{L} \mid T_{count\vee}(\alpha) = p\}$

Table 2: Definitions for some restricted languages.

Whilst for small examples, the total cost of using test functions may be greater than undertaking a classical consistency check, in general, they can be cost-effective when used to determine the probability of consistency as discussed in the next section.

3.3 Probability of inconsistency

Different types of restricted language that are finite, together with test functions, give us the apparatus to exploit probability theory for our notion of probable consistency. We use the usual definitions for probability theory, some of which are briefly revised below.

Definition 3.3 *A* **sample space** *is the set of all possible outcomes for an experiment. Here, we have used the set of formulae in a restricted language as a sample space. An* **observation** *is a subset of a sample space. Here we have used a test function to identify subsets of the sample space, and so we use test functions to define observations.*

Definition 3.4 *A* **probability function***, denoted P is a function from observations to the $[0, 1]$ interval such that: (1) for every observation X, $0 \leq P(X) \leq 1$; (2) for the sample space S, $P(S) = 1$; and (3) if A and B are exclusive observations, then $P(A \cup B) = P(A) + P(B)$. A* **conditional probability** *denoted $P(A \mid B)$ is $P(A \cap B)/P(B)$.*

Definition 3.5 *Let $\mathcal{L}_r$ be a restricted language. A* **uniform distribution** *for $\mathcal{L}_r$ is a probability function where if $\mathcal{L}_r$ contains n formulae, then for each α in $\mathcal{L}_r$, $P(\alpha) = 1/n$.*

We will assume a uniform distribution for every restricted language in this paper. Using a uniform distribution over each set $\mathcal{L}_r$, we can determine probabilities for subsets of $\mathcal{L}_r$ using the test function. We consider non-uniform distributions in Section 6.4.

Example 3.7 *For the $\mathcal{L}_{example2}$ restricted language (presented in Example 3.2), the proportion of inconsistent formulae is 1/3. Therefore, assuming the uniform distribution, the probability of inconsistency is 1/3.*

We can derive updated information using conditional probabilities such as in the following example.

Example 3.8 *Let $\mathcal{L}_{example5} \subseteq \mathcal{L}$ and let $\alpha \in \mathcal{L}_{example5}$. Assuming we are only using language $\mathcal{L}_{example5}$, we have a probability of any formula α being consistent. Let the proportion of inconsistent formulae in $\mathcal{L}_{example5}$ be 1/2, which can be represented by the following probability:*

$$P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_{example5}) = 1/2$$

Also assume there is a test function T_x from $\mathcal{L}_{example5}$ into $\{yes, no\}$. This test function can be used as a membership function for a subset of $\mathcal{L}_x$ of $\mathcal{L}_{example5}$. Suppose that the proportion $|\mathcal{L}_x|/|\mathcal{L}_{example5}|$ is 1/2. This we can represent by $P(\mathcal{L}_x \mid \mathcal{L}_{example5}) = 1/2$. Finally, assume the following joint probability:

$$P(\mathcal{L}_{inconsistent} \cap \mathcal{L}_x \cap \mathcal{L}_{example5}) = 1/5$$

Using these probabilities, we obtain the following conditional probability:

$$P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_x \cap \mathcal{L}_{example5}) = 2/5$$

This means that for the formula α, if $T_x(\alpha) = yes$, then the belief in α being inconsistent drops from 1/2 to 2/5.

Clearly, if $\alpha \in \mathcal{L}_r$ and $P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_r) = 1$ then $\{\alpha\} \vdash \perp$ holds. Similarly, if for all $\alpha \in \mathcal{L}_r$, $\{\alpha\} \vdash \perp$ holds, then $P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_r) = 1$.

Definition 3.6 *A* **probable consistency system** *is a set Π of conditional probability statements of the form $P(\mathcal{L}_{inconsistency} \mid \mathcal{L}_{r_i})$. Let Π be defined as follows,*

$$\Pi = \{P(\mathcal{L}_{inconsistency} \mid \mathcal{L}_{r_1}) = p_1, ..., P(\mathcal{L}_{inconsistency} \mid \mathcal{L}_{r_n}) = p_n\}.$$

The **universe** *for Π is $\mathcal{L}_{r_1} \cup ... \cup \mathcal{L}_{r_n}$, which is a subset of $\mathcal{L}$.*

In general, we may find by using a battery of test functions that a formula α is a member of a number of restricted languages. If we denoted these languages $\mathcal{L}_{x_1}, ..., \mathcal{L}_{x_n}$, then this can be captured by a conditional probability of the form:

$$P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_{x_1} \cap ... \cap \mathcal{L}_{x_n})$$

Couching test functions in terms of observations means that we can undertake a series of tests on a formula, and that the net result is independent of the order in which the tests are done. We consider the question of selecting good test functions in Section 6.3.

For a probable consistency system Π, and a formula α, the conditional probability chosen is the conditional probability with the most specific reference class for the test results.

Definition 3.7 *If Π is a probable consistency system, then Π_α and Π_α^{min} are subsets defined as follows.*

$$\Pi_\alpha = \{P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_i) \in \Pi \mid \alpha \in \mathcal{L}_i \textit{ according to some combination of test functions } \}$$

$$\Pi_\alpha^{min} = \{P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_i) \in \Pi_\alpha \mid \textit{ there is no } P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_j) \in \Pi_\alpha \textit{ such that } \mathcal{L}_i \subset \mathcal{L}_j\}$$

Definition 3.8 *Let Π be a probable consistency system, and let τ be a fixed threshold in the interval* $0.5 - 1.0$.

If $\Pi_\alpha^{min} = \{\mathcal{L}_{inconsistent} \mid \mathcal{L}_r\}$
and $\alpha \in \mathcal{L}_r$
and $P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_r) \geq \tau$,
then α is **probably inconsistent** *according to* Π.

The aim of using a probable consistency system Π with a formula α is to undertake enough tests to determine whether or not α is probably inconsistent. The threshold τ ranges from the very credulous (i.e. close to 0.5) to the very skeptical (i.e. close to 1.0). If $\tau = 1$, then we return to classical consistency.

Definition 3.9 *If Π is a probable consistency system, then* **probable consistency checking** *is a function $check_\Pi$ from $\mathcal{L}$ into $\{Yes, No\}$ such that for each $\alpha \in \mathcal{L}$, the following conditions hold.*

If α is probably inconsistent according to Π, then $check_\Pi(\alpha)$ = No

If α is not probably inconsistent according to Π, then $check_\Pi(\alpha)$ = Yes.

We consider some properties of probable consistency systems in Section 5.

3.4 Generating restricted languages

For each restricted language that we can identify using the test functions, we need to determine the cardinality of the language, and the proportion of formulae that are inconsistent. We have a number of options for this including:

- Listing every member explicitly and determining which individuals are inconsistent.
- Using a "scheme" to represent a class of formulae that can be systematically instantiated with atoms. A restricted language may be based on a set of such schema.
- Identifying general equations for determining the cardinality of a language and the proportion of inconsistent formulae that it contains.

Listing every formula in a restricted language is labourious for all but the smallest sets (unless we use automated techniques as discussed in Section 6.3). We therefore need to systematically determine the cardinality of restricted languages and the proportion of inconsistent formulae in a restricted language, for a given definition for the restricted language. Below, we consider schema in more detail. In Section 4, we consider some general equations for restricted languages.

Definition 3.10 *Let $\mathcal{V}$ be a set of meta-variables quantifying over some set of atoms $\mathcal{A}$. A* **scheme** *is formed from $\mathcal{V}$ and the usual logical symbols such that: (1) all elements of $\mathcal{V}$ are schema; (2) if X is a scheme, then $\neg X$ is a scheme; (3) if X and Y are schema, and $\diamond$ is a binary logical symbol (i.e. $\wedge$ or $\vee$), then $X \diamond Y$ is a scheme; and (4) in each scheme there is no repetition of a meta-variable.*

Example 3.9 *Let X and Y be meta-variables, then $\neg X$, $\neg\neg X$, $\neg\neg\neg X$,.... are schema. Also $X \wedge Y$ and $X \wedge \neg Y$ are schema. But $X \wedge X$ is not a scheme because a meta-variable is repeated.*

Definition 3.11 *Let $n \in \mathbb{N}$. An* **n-place scheme** *is a scheme composed of n meta-variables.*

Example 3.10 *Let X and Y be meta-variables, then $\neg X$, $\neg\neg X$, $\neg\neg\neg X$,.... are 1-place schema, and $X \wedge Y$, $\neg(X \wedge Y)$, and $X \wedge \neg Y$ are 2-place schema.*

Definition 3.12 *Let $\mathcal{A}$ be a set of atoms. A set of atomic n-tuples is the set $\mathcal{A}^n$.*

Definition 3.13 *Let $\mathcal{A}$ be a set of atoms. Let Θ be an n-place scheme, where X_1 is the first meta-variable in Θ (going from left to right) and,..., and X_n is the nth meta-variable in Θ. The language generated by the pair $(\mathcal{A}, \Theta)$ is the set of formulae formed by instantiating the scheme with each atomic n-tuple s.t. if $(\alpha_1, ..., \alpha_n)$ is an atomic n-tuple, then a formula is formed by substituting X_1 with α_1 and,..., and substituting X_n with α_n.*

Example 3.11 *Let $(X \wedge Y) \vee Z$ be a scheme and $\mathcal{A} = \{\alpha, \beta\}$ be a set of atoms. So the set of atomic 3-tuples is:*

$$\begin{array}{cc}
(\alpha,\alpha,\alpha) & (\alpha,\alpha,\beta) \\
(\alpha,\beta,\alpha) & (\alpha,\beta,\beta) \\
(\beta,\alpha,\alpha) & (\beta,\alpha,\beta) \\
(\beta,\beta,\alpha) & (\beta,\beta,\beta)
\end{array}$$

The language generated by $(\mathcal{A}, (X \wedge Y) \vee Z)$ is:

$$\begin{array}{cc}
((\alpha \wedge \alpha) \vee \alpha) & ((\alpha \wedge \alpha) \vee \beta) \\
((\alpha \wedge \beta) \vee \alpha) & ((\alpha \wedge \beta) \vee \beta) \\
((\beta \wedge \alpha) \vee \alpha) & ((\beta \wedge \alpha) \vee \beta) \\
((\beta \wedge \beta) \vee \alpha) & ((\beta \wedge \beta) \vee \beta)
\end{array}$$

We will use the concepts of schema and atomic n-tuples in the next section to help in determining the cardinality of useful restricted languages, and the proportion of inconsistent formulae in them.

4 Analysis of some restricted languages

In this section, we consider some examples of restricted languages in more detail.

4.1 Restricted language of conjunctions of atoms

We first consider the restricted language of conjunctions of atoms to illustrate the framework.

Definition 4.1 *Let $\mathcal{L}_{\wedge atoms}$ be the set of formulae defined as follows:*

$$\mathcal{L}_{\wedge atoms} = \{\alpha_1 \wedge .... \wedge \alpha_h \mid \alpha_1, ..., \alpha_h \in \mathcal{L}_{\mathcal{A}}\}$$

The formulae that are composed of h conjuncts, can be generated by the pair consisting of $\mathcal{L}_{atoms}$ and an h place scheme:

$$X_1 \wedge ... \wedge X_h$$

We can obtain the subset if $\mathcal{L}_{\wedge atoms}$ of formulae of a particular number of conjuncts, say h conjuncts, by taking the intersection of $\mathcal{L}_{\wedge atoms}$ with $\mathcal{L}_{count\wedge=p}$ and $h = p + 1$.

Proposition 4.1 *For the restricted languages $\mathcal{L}^{\mathcal{A}}_{\wedge atoms}$, and $\mathcal{L}_{count\wedge=p}$, we have the following cardinality equation, where $|\mathcal{A}| = k$ and $h = p + 1$:*

$$|\mathcal{L}^{\mathcal{A}}_{\wedge atoms} \cap \mathcal{L}_{count\wedge=p}| = k^h$$

Example 4.1 *Consider $\mathcal{A} = \{\alpha, \beta\}$, and $\mathcal{L}_{count\wedge=4}$. So $|\mathcal{L}^{\mathcal{A}}_{\wedge atoms} \cap \mathcal{L}_{count\wedge=4}|$ = 32.*

No formula in $\mathcal{L}_{\wedge atoms}$ is inconsistent, and so $\mathcal{L}_{\wedge atoms} \cap \mathcal{L}_{inconsistent} = \emptyset$.

4.2 Restricted language of clauses

Definition 4.2 *Let $\mathcal{L}_{clauses}$ be the largest subset of $\mathcal{L}$ where each $\alpha \in \mathcal{L}_{clauses}$ is a clause.*

Example 4.2 *The following schema $X \vee Y$, $\neg X \vee Y$, $X \vee \neg Y$, and $\neg X \vee \neg Y$ together with $\mathcal{L}_{atoms}$ generate the subset of $\mathcal{L}_{clauses}$ that is the set of binary clauses.*

Example 4.3 *Consider $\mathcal{A} = \{\alpha, \beta\}$. For this, the set $\mathcal{L}^{\mathcal{A}}_{clauses} \cap \mathcal{L}_{count\vee=1}$, is*

$$\begin{array}{cccc}
\alpha \vee \alpha & \alpha \vee \beta & \beta \vee \alpha & \beta \vee \beta \\
\alpha \vee \neg\alpha & \alpha \vee \neg\beta & \beta \vee \neg\alpha & \beta \vee \neg\beta \\
\neg\alpha \vee \alpha & \neg\alpha \vee \beta & \neg\beta \vee \alpha & \neg\beta \vee \beta \\
\neg\alpha \vee \neg\alpha & \neg\alpha \vee \neg\beta & \neg\beta \vee \neg\alpha & \neg\beta \vee \neg\beta
\end{array}$$

Proposition 4.2 *For the restricted languages $\mathcal{L}^{\mathcal{A}}_{clauses}$, and $\mathcal{L}_{count\vee=p}$, we have the following cardinality equation, where $|\mathcal{A}| = k$ and h = p+1:*

$$|\mathcal{L}^{\mathcal{A}}_{clauses} \cap \mathcal{L}_{count\vee=p}| = 2^h \times k^h$$

Proof: Consider generating the restricted language using schema. The set of atomic n-tuples has cardinality k^h, so for each scheme, there are k^h generated formulae, and the set of schema has cardinality 2^h. □

No formula in $\mathcal{L}_{clauses}$ is inconsistent, and so $\mathcal{L}_{clauses} \cap \mathcal{L}_{inconsistent} = \emptyset$.

4.3 Restricted language of conjunctions of literals

Definition 4.3 *Let $\mathcal{L}_{\wedge literals}$ be the set of formulae defined as follows:*

$$\mathcal{L}_{\wedge literals} = \{\alpha_1 \wedge .... \wedge \alpha_h \mid \alpha_1, ..., \alpha_h \in \mathcal{L}_{literals}\}$$

Number of conjuncts	Number of formulae	Number of consistent formulae	Proportion of consistent formulae
1	2	2	1
2	4	2	1/2
3	8	2	1/4
4	16	2	1/8
5	32	2	1/16
6	64	2	1/32
7	128	2	1/64
8	256	2	1/128

Table 3: Proportion of consistent formulae for $\mathcal{L}^{\mathcal{A}}_{\wedge literals}$ where $|\mathcal{A}| = 1$. The values are obtained using Proposition 4.8.

Number of conjuncts	Number of formulae	Number of consistent formulae	Proportion of consistent formulae
1	4	4	1.00
2	16	12	0.75
3	64	28	0.43
4	256	60	0.23
5	1024	124	0.12
6	4096	252	0.06
7	16384	508	0.03
8	65536	1020	0.01

Table 4: Proportion of consistent formulae for $\mathcal{L}^{\mathcal{A}}_{\wedge literals}$ where $|\mathcal{A}| = 2$. The values given to two decimal places are obtained using Proposition 4.8.

Example 4.4 *The following schema $X \wedge Y$, $\neg X \wedge Y$, $X \wedge \neg Y$, and $\neg X \wedge \neg Y$ together with $\mathcal{L}_{atoms}$ generates the subset of $\mathcal{L}_{\wedge literals}$ that is the set of binary conjuncts.*

Example 4.5 *Consider $\mathcal{A} = \{\alpha, \beta\}$. For this, the set $\mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=1}$, is*

$$\begin{array}{cccc}
\alpha \wedge \alpha & \alpha \wedge \beta & \beta \wedge \alpha & \beta \wedge \beta \\
\alpha \wedge \neg\alpha & \alpha \wedge \neg\beta & \beta \wedge \neg\alpha & \beta \wedge \neg\beta \\
\neg\alpha \wedge \alpha & \neg\alpha \wedge \beta & \neg\beta \wedge \alpha & \neg\beta \wedge \beta \\
\neg\alpha \wedge \neg\alpha & \neg\alpha \wedge \neg\beta & \neg\beta \wedge \neg\alpha & \neg\beta \wedge \neg\beta
\end{array}$$

Proposition 4.3 *For the restricted languages $\mathcal{L}^{\mathcal{A}}_{\wedge literals}$, and $\mathcal{L}_{count\wedge=p}$, we have the following cardinality equation, where $|\mathcal{A}| = k$ and h = p+1:*

$$|\mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=p}| = 2^h \times k^h$$

Proof: Consider generating the restricted language using schema. The set of atomic h-tuples has cardinality k^h, so for each scheme, there are k^h generated formulae, and the set of schema has cardinality 2^h. $\Box$

Number of conjuncts	Number of formulae	Number of consistent formulae	Proportion of consistent formulae
1	6	6	1.00
2	36	30	0.83
3	216	126	0.53
4	1296	462	0.35
5	7776	1566	0.20
6	46656	5310	0.11
7	279936	15966	0.05
8	1679616	49422	0.02

Table 5: Proportion of consistent formulae for $\mathcal{L}^{\mathcal{A}}_{\wedge literals}$ where $|\mathcal{A}| = 3$. The values given to two decimal places are obtained using Proposition 4.8.

We now consider a way to determine the proportion of $\mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=p}$ that are inconsistent, for any $\mathcal{A}$ and p.

Definition 4.4 *Each $\Phi \in \wp(\mathcal{L}^{\mathcal{A}}_{literals})$ can be used to generate a subset of $\mathcal{L}^{\mathcal{A}}_{\wedge literals}$. Let $h \in N^+$, $con(\Phi, h) = \{\phi_1 \wedge ... \wedge \phi_h \mid \phi_1, .., \phi_h \in \Phi\}$.*

Clearly, $con(\Phi, h) \subseteq \mathcal{L}^{\mathcal{A}}_{\wedge literals}$. Indeed, if we let $\Phi = \mathcal{L}^{\mathcal{A}}_{literals}$ and $\wp(\Phi) = \{\Phi_1, .., \Phi_k\}$, where $k = 2^{|\Phi|}$, and let $h = p+1$, then $con(\Phi_1, h) \cup .. \cup con(\Phi_k, h) = \mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=p}$. However, $con(\Phi_1, h), .., con(\Phi_k, h)$ is not a partitioning of $\mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=p}$ as illustrated by the following example.

Example 4.6 *Let $\Phi = \{\alpha, \neg\alpha\}$. So $\wp(\Phi) = \{\{\alpha\}, \{\neg\alpha\}, \{\alpha, \neg\alpha\}, \{\}\}$.*

$$\begin{array}{ll} con(\{\alpha\}, 2) & = \{\alpha \wedge \alpha\} \\ con(\{\neg\alpha\}, 2) & = \{\neg\alpha \wedge \neg\alpha\} \\ con(\{\alpha, \neg\alpha\}, 2) & = \{\alpha \wedge \alpha, \alpha \wedge \neg\alpha, \neg\alpha \wedge \alpha, \neg\alpha \wedge \neg\alpha, \} \end{array}$$

In order to partition $\mathcal{L}^{\mathcal{A}}_{literals} \cap \mathcal{L}_{count\wedge=p}$, we need to subtract from $con(\Phi_i, h)$, the formulae that occur in $con(\Phi_j, h)$, where $\Phi_j \subset \Phi_i$. The subset of $con(\Phi_i, h)$ where the formulae in $con(\Phi_j, h)$ for all $\Phi_j \subset \Phi_i$ have been subtracted is denoted $con^*(\Phi_i, h)$.

Definition 4.5 *The set of strict subsets of Φ is denoted $\mathcal{S}(\Phi)$ and defined as follows:*

$$\mathcal{S}(\Phi) = \{\Psi \in \wp(\Phi) \mid \Psi \subset \Phi\}$$

Definition 4.6 *Let $\Phi \in \wp(\mathcal{L}^{\mathcal{A}}_{literals})$ and $h \in \mathbb{N}$. Let $con^*(\Phi, h)$ be defined as follows, where $\mathcal{S}(\Phi) = \{\Psi_1, .., \Psi_r\}$ and $r = 2^{|\Phi|} - 1$.*

$$con^*(\Phi, h) = con(\Phi, h) - (con(\Psi_1, h) \cup ... \cup con(\Psi_r, h))$$

Clearly, for any h, we have $con^*(\{\}, h) = \emptyset$. Furthermore, if we let $\Phi = \mathcal{L}^{\mathcal{A}}_{literals}$ and $\wp(\Phi) = \{\Phi_1, .., \Phi_j\}$ and $h = p + 1$, then $con^*(\Phi_1, h), .., con^*(\Phi_j, h)$ is a partitioning of $\mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=p}$.

Definition 4.7 *Let the* **consubsets function***, and the* **inconsubsets function***, denoted consubsets and inconsubsets respectively, be functions from $\wp(\mathcal{L})$ into $\wp(\wp(\mathcal{L}))$ defined as follows, where $\Gamma \in \wp(\mathcal{L})$.*

$$consubsets(\Gamma) = \{\Phi \in \wp(\Gamma) \mid \Phi \not\vdash \perp\}$$

$$inconsubsets(\Gamma) = \{\Phi \in \wp(\Gamma) \mid \Phi \vdash \bot\}$$

Proposition 4.4 *Let* $\Phi = \mathcal{L}^{\mathcal{A}}_{literals}$ *and let* $consubsets(\Phi) = \{\Phi_1, .., \Phi_x\}$ *and* $inconsubsets(\Phi) = \{\Psi_1, .., \Psi_y\}$ *and* $h = p + 1$.

$$con^*(\Phi_1, h) \cup .. \cup con^*(\Phi_x, h) = \mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=p} \cap \mathcal{L}_{consistent}$$

$$con^*(\Psi_1, h) \cup .. \cup con^*(\Psi_y, h) = \mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=p} \cap \mathcal{L}_{inconsistent}$$

Example 4.7 *Let* $\Phi = \{\alpha, \neg\alpha, \beta, \neg\beta\}$, $\wp(\Phi)$ *is the following 16 subsets.*

$$\begin{array}{c}
\Phi_1 = \{\},\ \Phi_2 = \{\alpha\},\ \Phi_3 = \{\neg\alpha\},\ \Phi_4 = \{\beta\},\ \Phi_5 = \{\neg\beta\}, \\
\Phi_6 = \{\alpha, \beta\},\ \Phi_7 = \{\alpha, \neg\alpha\},\ \Phi_8 = \{\beta, \neg\beta\},\ \Phi_9 = \{\alpha, \neg\beta\},\ \Phi_{10} = \{\beta, \neg\alpha\},\ \Phi_{11} = \{\neg\alpha, \neg\beta\}, \\
\Phi_{12} = \{\alpha, \beta, \neg\alpha\},\ \Phi_{13} = \{\alpha, \beta, \neg\beta\},\ \Phi_{14} = \{\alpha, \neg\alpha, \neg\beta\},\ \Phi_{15} = \{\beta, \neg\alpha, \neg\beta\}, \\
\Phi_{16} = \{\alpha, \beta, \neg\alpha, \neg\beta\}
\end{array}$$

where $inconsubsets(\Phi) = \{\Phi_7, \Phi_8, \Phi_{12}, \Phi_{13}, \Phi_{14}, \Phi_{15}, \Phi_{16}\}$

Now consider $h = 3$,

$$\begin{array}{c}
|con(\Phi_2, 3)| = |con(\Phi_3, 3)| = |con(\Phi_4, 3)| = |con(\Phi_5, 3)| = 1 \\
|con^*(\Phi_2, 3)| = |con^*(\Phi_3, 3)| = |con^*(\Phi_4, 3)| = |con^*(\Phi_5, 3)| = 1
\end{array}$$

$$\begin{array}{c}
|con(\Phi_6, 3)| = |con(\Phi_7, 3)| = |con(\Phi_8, 3)| = |con(\Phi_9, 3)| = |con(\Phi_{10}, 3)| = |con(\Phi_{11}, 3)| = 8 \\
|con^*(\Phi_6, 3)| = |con^*(\Phi_7, 3)| = |con^*(\Phi_8, 3)| = |con^*(\Phi_9, 3)| = |con^*(\Phi_{10}, 3)| = |con^*(\Phi_{11}, 3)| = 6
\end{array}$$

$$\begin{array}{c}
|con(\Phi_{12}, 3)| = |con(\Phi_{13}, 3)| = |con(\Phi_{14}, 3)| = |con(\Phi_{15}, 3)| = 27 \\
|con^*(\Phi_{12}, 3)| = |con^*(\Phi_{13}, 3)| = |con^*(\Phi_{14}, 3)| = |con^*(\Phi_{15}, 3)| = 6
\end{array}$$

$$\begin{array}{c}
|con(\Phi_{16}, 3)| = 64 \\
|con^*(\Phi_{16}, 3)| = 0
\end{array}$$

So the number of inconsistent formulae is $|con^*(\Phi_7, 3)| + |con^*(\Phi_8, 3)| + |con^*(\Phi_{12}, 3)| + |con^*(\Phi_{13}, 3)| + |con^*(\Phi_{14}, 3)| + |con^*(\Phi_{15}, 3)| + |con^*(\Phi_{16}, 3)| = 36$.

Next, we recall the standard definition for determing the number of subsets of cardinality i that can be formed from a set of cardinality k.

Definition 4.8 *For* $k, i \in \mathbb{N}$, *where* $i \leq k$,

$$combinations(k, i) = \frac{k!}{i! \times (k - i)!}$$

We also recall the standard definition for multinomial coefficient for determining the number of permutations of length h that can be formed from a set Φ of cardinality i with repetitions from that set being allowed. For the definition of *coefficient*(Z) below, Z is a list of the form $< x_1, ..., x_i >$ where x_1 denotes the number of repetitions of the first item in Φ and, ..., and x_i denotes the number of repetitions of the ith item in Φ.

Definition 4.9 *For $i, h \in \mathbb{N}^+$,*

$$\textit{coefficient}(< x_1, ..., x_i >) = \frac{h!}{x_1! \times ... \times x_i!}$$

where $x_1, .., x_i \in \mathbb{N}^+$ and $x_1 + ... + x_i = h$.

Using this definition, we define $multinomials(i, h)$ as the sum of all possible multinomial coefficients that can be formed given i and h.

Definition 4.10 *For $i, h \in \mathbb{N}^+$, where $i \leq h$,*

$$multinomials(i, h) = \textit{coefficient}(Z_1) + ... + \textit{coefficient}(Z_s)$$

where $\{Z_1, ..., Z_s\}$ is defined as the following set

$$\{ < x_1, ..., x_i > \mid x_1 + ... + x_i = h \textit{ and } x_1 \in \mathbb{N}^+ \textit{ and ... and } x_i \in \mathbb{N}^+ \}$$

For $i, h \in \mathbb{N}^+$, where $i > h$,

$$multinomials(i, h) = 0$$

Proposition 4.5 *For $\Phi \in \wp(\mathcal{L}^{\mathcal{A}}_{literals})$, and $i, h \in \mathbb{N}$, where $|\Phi| = i$,*

$$|con^*(\Phi, h)| = multinomials(i, h)$$

Proof: Every literal in Φ appears at least once in each formula in $con^*(\Phi, h)$. Since each literal in Φ occurs at least once in each formula, no formula in $con^*(\Phi, h)$ occurs in any set $con^*(\Psi, h)$ where $\Psi \subset \Phi$. The number of occurrences of each literal in each formula can be represented by a tuple $< x_1, .., x_i >$ where $x_1, .., x_i \in \mathbb{N}$. For this, the sum $x_1 + .. + x_i = h$. More than one formula may correspond to the same tuple. The number of formulae that corresponds to each tuple, is given by the multinomial coefficient *coefficient*$(< x_1, ..., x_i >)$. The set of all possible tuples for $con^*(\Phi, h)$ where $|\Phi| = i$, is $\{< x_1, ..., x_i >|$ $x_1 + ... + x_i = h$ and $x_1, .., x_i \in \mathbb{N}^+\}$, and so the sum of the number of formulae that correspond to each tuple is given by $multinomials(i, h)$. □

Definition 4.11 *Let $\mathcal{A}$ be any set of atoms and let $consubset(\mathcal{L}^{\mathcal{A}}_{literals}, i) \subseteq consubset(\mathcal{L}^{\mathcal{A}}_{literals})$ such that each set in $consubset(\mathcal{L}^{\mathcal{A}}_{literals}, i)$ is of cardinality i.*

$$consubset(\mathcal{L}^{\mathcal{A}}_{literals}, i) = \{ \Phi \in consubset(\mathcal{L}^{\mathcal{A}}_{literals}) \mid \ |\Phi| = i \}$$

Proposition 4.6 *For $|\mathcal{A}| = k$ and $i \leq k$,*

$$|consubset(\mathcal{L}^{\mathcal{A}}_{literals}, i)| = combinations(k, i) \times 2^i$$

Proof: Since each consistent subset Φ of $\mathcal{L}^{\mathcal{A}}_{literals}$ is such that for each literal $\alpha \in \Phi$, the complement of α is not in Φ. So for each subset Ψ of $\mathcal{A}$ of cardinality i, all the consistent subsets of $\mathcal{A}$ can be formed by taking all combinations of adding negation to the atoms in Ψ. So each subset of $\mathcal{A}$ of cardinality spawns 2^i consistent subsets of $\mathcal{L}^{\mathcal{A}}_{literals}$. Since there are $combinations(k, i)$ subsets of $\mathcal{A}$ of cardinality i, we obtain the result. □

In the following definition, $count(k, i, h)$ is a function that gives the sum of the number of consistent conjunctions of literals, composed of h conjuncts, that can be formed from each subset of Γ of cardinality i of $\mathcal{L}^{\mathcal{A}}_{literals}$ and not formed from any subset of Γ. We formalize this below, where $|\mathcal{A}| = k$.

Definition 4.12 *For $k = |\mathcal{A}|$, and $h \in \mathbb{N}^+$, and $consubset(\mathcal{L}^{\mathcal{A}}_{literals}, i) = \{\Phi_1, ..., \Phi_j\}$.*

$$count(k, i, h) = |con^*(\Phi_1, h)| + ... + |con^*(\Phi_j, h)|$$

Proposition 4.7 *For $|\mathcal{A}| = k$ and $i \leq k$.*

$$count(k, i, h) = combinations(k, i) \times 2^i \times multinomials(i, h)$$

Proof: From Proposition 4.6, we have the first two factors. The third factor is from Proposition 4.5. □

Proposition 4.8 *The number of consistent formulae in $\mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=p}$ is count(k,h),calculated as follows, where $k = |\mathcal{A}|$, and h = p+1.*

$$count(k, h) = \sum_{i=1}^{k} count(k, i, h)$$

Proof: Let $consubset(\mathcal{L}^{\mathcal{A}}_{literals}) = \{\Phi_1, .., \Phi_x\}$, and so by Proposition 4.4,

$$\begin{aligned} count(k, h) &= |con^*(\Phi_1, h) \cup ... \cup con^*(\Phi_x, h)| \\ &= |con^*(\Phi_1, h)| + ... + |con^*(\Phi_x, h)| \end{aligned}$$

Since $consubset(\mathcal{L}^{\mathcal{A}}_{literals}) = \{\Phi_1, .., \Phi_x\}$ = $\bigcup_{i=1}^{k} consubset(\mathcal{L}^{\mathcal{A}}_{literals}, i)$ and

for all $v, w \leq k$, if $v \neq w$, then $consubset(\mathcal{L}^{\mathcal{A}}_{literals}, v) \cap consubset(\mathcal{L}^{\mathcal{A}}_{literals}, w) = \emptyset$

we have $count(k, h)$ = $count(k, 1, h)$ + ... + $count(k, k, h)$.□

For $|\mathcal{A}| = k$, and h = p+1, the proportion of consistent formulae in $\mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=p}$ is:

$$\frac{count(k, h)}{2^h \times k^h}$$

where the denominator is from Proposition 4.3 and the numerator is from Proposition 4.8.

Example 4.8 *Let $|\mathcal{A}| = 3$ and $h = 2$.*

$$multinomials(1, 2) = \frac{2!}{2!} = 1$$

$$multinomials(2, 2) = \frac{2!}{1!1!} = 2$$

$$\begin{aligned} count(3, 1, 2) &= combinations(3, 1) \times 2^1 \times multinomials(1, 2) = 3 \times 2 \times 1 = 6 \\ count(3, 2, 2) &= combinations(3, 2) \times 2^2 \times multinomials(2, 2) = 3 \times 4 \times 2 = 24 \\ count(3, 3, 2) &= combinations(3, 3) \times 2^3 \times multinomials(3, 2) = 1 \times 8 \times 0 = 0 \end{aligned}$$

$$count(3, 2) = count(3, 1, 2) + count(3, 2, 2) + count(3, 3, 2) = 6 + 24 + 0 = 30$$

So $P(\mathcal{L}_{consistent} \mid \mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=1})$ = 30/36.

Example 4.9 *Let* $|\mathcal{A}| = 2$*, so* $P(\mathcal{L}_{inconsistent} \mid \mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=3})$ *= 9/16.*

Further examples are given in Table 3 to Table 5.

Couching Proposition 4.8 in terms of multinomial coefficients opens the opportunity for incorporating probabilities over literals using multinomial distributions.

Now we consider a special case of Proposition 4.8 where the conjunctions are binary.

Proposition 4.9 *The proportion of consistent formulae in* $\mathcal{L}^{\mathcal{A}}_{\wedge literals} \cap \mathcal{L}_{count\wedge=1}$ *is* $1 - 1/(2k)$*, where* k = $|\mathcal{A}|$.

Proof: We have

$$\begin{aligned} count(k,1,2) &= combinations(k,1) \times 2^1 \times multinomials(1,2) \\ &= k!/(1!(k-1)!) \times 2 \times 1 = 2k \end{aligned}$$

$$\begin{aligned} count(k,2,2) &= combinations(k,2) \times 2^2 \times multinomials(2,2) \\ &= k!/(2!(k-2)!) \times 4 \times 2 = 4k(k-1) \end{aligned}$$

So $count(k,2)$ = $count(k,1,2)+count(k,2,2)$ = $2k+4k(k-1)$, and the proportion of consistent formulae is:

$$\frac{2k + 4k(k-1)}{4k^2} = 1 - \frac{1}{2k}$$

4.4 Restricted language of formulae in CNF

Definition 4.13 *Let* $\mathcal{L}_{CNF}$ *be the largest subset of* $\mathcal{L}$ *where each* $\alpha \in \mathcal{L}_{CNF}$ *is in CNF.*

The language $\mathcal{L}_{CNF}$ contains $\mathcal{L}_{\wedge literals}$ and $\mathcal{L}_{clauses}$. Testing whether formula α is in $\mathcal{L}_{CNF}$ can be efficiently determined by a very simple finite state machine.

Proposition 4.10 *For the restricted languages* $\mathcal{L}^{\mathcal{A}}_{CNF}$*,* $\mathcal{L}_{count\wedge=p}$*, and* $\mathcal{L}_{count\vee=q}$*, we have the following cardinality equation, where* $|\mathcal{A}| = k$ *and* $r = (p+q+1)$*:*

$$|\mathcal{L}^{\mathcal{A}}_{CNF} \cap \mathcal{L}_{count\wedge=p} \cap \mathcal{L}_{count\vee=q}| = 2^r \times k^r \times \frac{(p+q-1)!}{q!(p-1)!}$$

Proof: The first term represents all possible permutations of positive and negative literals for a formula composed of r literals. The second term is the cardinality of the set of atomic r-tuples, so for each scheme there are k^r generated formulae. The third term represents the number of ways of distributing q instances of the $\vee$ symbol between p instances of the $\wedge$ symbol, where for each $\wedge$ symbol, there may be zero or more $\vee$ symbols. So the product of the first and third terms gives the cardinality of the set of schema. □

We now focus on a very simple sublanguage of $\mathcal{L}_{CNF}$.

Proposition 4.11 *The proportion of inconsistent formulae in* $\mathcal{L}^{\mathcal{A}}_{CNF} \cap \mathcal{L}_{count\wedge=1} \cap \mathcal{L}_{count\vee=q}$ *is* $(1/2k)^{q+1}$*, where* $|\mathcal{A}| = k$*.*

Proof: Each formula $\alpha \in \mathcal{L}^{\mathcal{A}}_{CNF} \cap \mathcal{L}_{count\wedge=1} \cap \mathcal{L}_{count\vee=q}$ is a conjunction of two clauses. Let us denote α as $X_1 \vee .. \vee X_n \wedge Y_1 \vee .. \vee Y_m$. The formula is contradictory iff X_1 = ... = X_n and Y_1 = ... = Y_m and X_1 is the complement of Y_1. The probability that X_1 is the complement if Y_1 is $1/2k$. The probability that X_1 = ... = X_n and Y_1 = ... = Y_m is $(1/2k)^q$. Hence, the probability of α being inconsistent is $(1/2k) \times (1/2k)^q$.□

Similar results can be obtained for restricted languages such as $\mathcal{L}^{\mathcal{A}}_{CNF} \cap \mathcal{L}_{count\wedge=2} \cap \mathcal{L}_{count\vee=2}$, and $\mathcal{L}^{\mathcal{A}}_{CNF} \cap \mathcal{L}_{count\wedge=3} \cap \mathcal{L}_{count\vee=2}$.

5 Viability of probable consistency checking

Even though consistency checking for classical propositional logic is NP-complete, it is undertaken with some success in various ways in artificial intelligence, data and knowledge engineering, and software engineering, as discussed in Section 7. In addition, there are implemented consistency checking and model building systems for full classical propositional logic (for example [CZ93, BCP94, Sch94, CP95, Sla96]), for the incomplete GSAT-like systems for clausal classical propositional logic [SLM92, GW93, GM94, MSG97], and for the linear-time algorithms for propositional Horn formulae [DG94]. In this light, we consider some of the advantages of probable consistency checking in this section.

In classical consistency checking, if Δ is declared consistent, then it is classically consistent, whereas in probable consistency checking, if Δ is declared consistent with a high probability, then it is not necessarily classically consistent. However, classical consistency checking is not guaranteed to make a decision in polynomial time, whereas probable consistency checking is guaranteed to make a decision in polynomial time, if done with polynomial time tests. Unlike some approaches to viable consistency checking, probable consistency checking is not restricted to a subset of the classical propositional language, and it is complete in the sense that it always returns an answer — either "yes" or "no" — it never returns "don't know". Finally, the tests in probable consistency checking gives useful information about the nature of a putative inconsistency that can be used in inconsistency management.

In the following proposition, the first restriction ensures that each conditional probability in Π is not too expensive to use and the second ensures that the size of Π is not too expensive to store and search.

Proposition 5.1 *Probable consistency checking* $check_\Pi$ *is a polynomial time decision problem if* Π *is a probable consistency system where (1) for each* $P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_r) \in \Pi$*, the membership function for* $\mathcal{L}_r$ *is a polynomial time function, and (2) if* $\mathcal{L}_r = \mathcal{L}^{\mathcal{A}} \cap \mathcal{L}_{count\vee=x} \cap \mathcal{L}_{count\wedge=y} \cap \mathcal{L}_{count\neg=z}$*, then* $check_\Pi$ *is a polynomial function of the* $(|\Pi|, x, y, z)$ *tuple.*

Proof: Each $P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_r) \in \Pi$ is the probability of inconsistency conditioned on a set $\mathcal{L}_r$. Let the conditional probability with the conditioning set with the most expensive membership function, take time t for the membership check for α. By condition 1, t is a polynomial function of α. In the most inefficient searching of Π, the membership function for each conditioning set for each element of Π is applied to α. According to condition (2), $|\Pi|$ does not vary with α. Let $|\Pi|$ = s. So in the most inefficient case, $check_\Pi(\alpha)$ takes time $t \times s$. □.

Whilst consistency checking is a polynomial time decision problem for some $\alpha \in \mathcal{L}$, for example any $\alpha \in \mathcal{L}_{\wedge literals}$, it is important to support probable consistency checking for these formulae because (1) probable consistency checking provides a potentially useful abstraction of consistency checking, and (2) we require a uniform framework for probable consistency checking for a significant subset $\mathcal{L}_r$ of $\mathcal{L}$, even though consistency checking for some subsets of $\mathcal{L}_r$ can be done in polynomial time. In effect, we are trading accuracy for efficiency.

In probable consistency checking, accuracy is relative. As more tests are used on a formula, the smaller the restricted language it is a member of. A smaller conditioning restricted language means a more accurate conditional probability is chosen. If the conditioning is restricted to just the singleton set, then we have the following result.

Proposition 5.2 *For $\alpha \in \mathcal{L}$, if $\mathcal{L}_r = \{\alpha\}$, then $P(\mathcal{L}_{consistent} \mid \mathcal{L}_r) = 0$ or $P(\mathcal{L}_{consistent} \mid \mathcal{L}_r) = 1$.*

Whilst probable consisitency checking is not NP-complete (if only polynomial time test functions are used), there are costs inccurred: These costs include:

- Cost of calculating the conditional probabilities.
- Cost of storing the conditional probabilites.
- Cost of undertaking tests on a formulae to determine appropriate subset.

The following result shows one extreme of the trade-off between accuracy in probable consistency checking and the cost of storage.

Proposition 5.3 *Let Π be a probable consistency system. If for every $\alpha \in \mathcal{L}_r$, there is a probability statement of the form $P(\mathcal{L}_{consistent} \mid \{\alpha\}) = 1$ or $P(\mathcal{L}_{consistent} \mid \{\alpha\}) = 1$, then $|\Pi|$ = $|\mathcal{L}_r|$.*

Another trade-off is between accuracy and the cost of testing. In the result below, we see one extreme of this trade-off.

Proposition 5.4 *There is a test function T that for any $\alpha \in \mathcal{L}$, T identifies a subset $\mathcal{L}_r \subset \mathcal{L}$ such that $\alpha \in \mathcal{L}_r$, and either $P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_r) = 1$ holds or $P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_r) = 0$ holds. If each $\mathcal{L}_r$ is not a singleton set, then T is NP-complete.*

Proof: By contradiction. Suppose for any $\alpha \in \mathcal{L}$, there is a polynomial time membership test function T for the set $\mathcal{L}_r$ and either $P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_r) = 1$ or $P(\mathcal{L}_{inconsistent} \mid \mathcal{L}_r) = 0$. Since, $\alpha \in \mathcal{L}_{inconsistent}$ iff $\alpha \vdash \perp$, we have a polynomial time operation for consistency checking. But, this contradicts the NP-completeness property of classical consistency checking.$\Box$

The benefit of using a probable consistency system comes from repeated use of the store of conditional probabilities. One way of viewing a probability consistency statement is that it is a partial evaluation of a formula — where a complete evaluation is represented in Proposition 5.3. In this way, probable consistency checking is an abstraction of consistency checking. However, there is a cost involved with generating a probable consistency system, as can be seen from the results in Section 4. However, once determined they can be used repeatedly.

Given the difficulties of transfinite arithmetic, we cannot identify useful conditional probability statements for all sets of formulae. Consider $\mathcal{L}$, an interesting question is what is the proportion of inconsistent formulae, ie. what is $|\mathcal{L}_{inconsistent}|/|\mathcal{L}|$? But since $|\mathcal{L}_{inconsistent}|$ = $|\mathcal{L}|$ = $\aleph$, we either regard $|\mathcal{L}_{inconsistent}|/|\mathcal{L}|$ as undefined or there are many answers with the answer set being $\mathbb{N}$.

6 Applications issues

In this section, we consider how we may use a probable consistency system to highlight potentially problematical information. We then consider three developments of the framework for applications: (1) Building a library of probable consistency systems; (2) Selection of good test functions in a probable consistency system; and (3) Use of non-uniform distributions over formulae. Finally, we consider using probable consistency checking as the basis of an analytical framework to complement empirical approaches to consistency checking.

6.1 Highlighting potentially problematical information

In order to indicate how we may use probable consistency checking, we consider applications in monitoring requirements in negotiations and monitoring data streams.

Monitoring requirements in negotiations Negotiations often start from a position of conflict. A group of participants may each have their own perspectives and agenda, and the group as a whole has to move to a position that is consistent. Suppose, each participant represents their perspective by a classical logic formula, then the group view can be represented by a set of formulae (for example in requirements engineering [FGH+94, HN98]). If this set of formulae is inconsistent, then each participant is able to revise their contribution. This revision may be iterated a number of times until hopefully, a consistent set of formulae is eventually obtained. Now all this calls for a lot of consistency checking. Obviously each participant should put forward a consistent formula that reflects their requirements. Furthermore, the more constrained each participant's requirements are, the more difficult it is to find a solution for the group as a whole. Probable consistency checking provides a measure of how complex a participant's requirements are. Even if someone's requirements are consistent, discovering that the requirements are in a class where a high proportion of the formulae are inconsistent is an indicator of the complexity of the formula. Participants in a negotiation should therefore be encouraged to choose requirements that are represented by formulae from restricted languages that have a lower proportion of inconsistent members. Furthermore, with each cycle of negotiation, each participant should be encouraged to choose requirements that are represented by formulae from restricted languages that have a lower proportion of inconsistent members, otherwise there is a danger that the negotiations will move further away from a negotiated consensus.

Monitoring data streams Data streams are characterized by many items of information of a regular format, and content type, being sent at high frequency. Consider for example a newsfeed of business news reports where each news report is a small item of XML with the text entries restricted to individual words, numbers, or very short phrases, capturing company names, names of people, product names, and financial figures. Given these restrictions on the XML, we can straightforwardly translate each report into a logical formula without recourse to natural language processing. Once represented in logic, the news can be analysed logically. For example, reports from multiple sources can be integrated into a single report [Hun00a, Hun02a, Hun02c], and ramifications of news can be identified using logical argumentation [Hun00b, BH01, Hun01]. Probable consistency checking offers quick and cheap filtering of potentially problematical news reports when conjoined with integrity constraints and background facts. These problematical news reports can either be rejected or be further analysed. Further analysis may for example reveal that a particular source of news reports regularly have reports filtered out which in turn may lead to investigations into that source. The need for quick and cheap filtering of problematic data is replicated in other data stream situations such as database updates in very large data repositories and groupware messaging systems.

Probable consistency checking may also be useful in SAT problems, (eg SAT planning or diagnosis), we may again view the formulae being consider for a plan or diagnosis as arriving via a data stream, and use probable consistency checking to filter out individual formulae that are highly likely to be problematical. These can either be rejected or further analysed. Since SAT algorithms do not provide feedback on causes for a set of formulae being unsatisfiable, we may use SAT algorithms and probable consistency checking as complementary techniques, as discussed in Section 6.5.

6.2 Building a library of probable consistency systems

In order to construct a probable consistency system, we need to select a language $\mathcal{L}_r$, as the universe for it, and then consider particular subsets of $\mathcal{L}_r$ that we can identify using simple polynomial time tests. We have already assumed (see Definition 3.6) that each universe is finite, that different tests on $\mathcal{L}_r$ give us some of the subsets of $\mathcal{L}_r$, and that we need to determine the proportion of formulae that are inconsistent in each of these subsets of $\mathcal{L}_r$.

In this way, we can identify a series of universes $\mathcal{L}_{r_1}$, ... , $\mathcal{L}_{r_k}$, construct a probable consistency system for each of them, and then use the resulting set of probable consistency systems as a library. The intention is that a library can be used in different applications, and the work involved in constructing the library can be amortized over different applications over time.

To use the library, we select a probable consistency system with a universe that is "big enough" to contain the formulae used in the application. By big enough, we mean that there are at least the same number of propositional atoms in the universe as used in the application formulae and the complexity of the formulae in the application language is reflected in the universe. The choice of atom symbols used in a universe is not relevant. The reason we do not worry about the choice of atom symbols is that we can translate each atom symbol in the formulae of the application to an atom symbol in the universe by a mapping that renders the application language $\mathcal{L}_a$ isomorphic to a subset of the universe $\mathcal{L}_r$. We do not need to actually identify this isomorphic subset. All we need to know is that it exists. Though we do need to know the mapping from the atom symbols of $\mathcal{L}_a$ to the atom symbols of $\mathcal{L}_r$.

The utility of a library therefore depends on: (1) having enough probable consistency systems such that at least one has a universe that contains a language isomorphic to the application language; (2) it being relatively efficient to check which universe to consider for an application formula by using a simple classification test and finding an appropriate isomorphism; and (3) each probable consistency system has enough tests to determine for each $\alpha \in \mathcal{L}_r$ (where $\mathcal{L}_r$ is the universe of the probable consistency checking system) whether α is probably consistent or probably inconsistent.

These criteria call for a balance to be struck between having larger universes and having smaller universes. An extreme example of a library would be where each universe is a singleton set. If the formula being tested is isomorphic to the element in the universe, then the probability of consistency is either 0 or 1. Clearly this would be untenable (see also Propositions 5.3 and 5.4). In contrast, if we have very large universes, then it may be necessary to undertake a large number of tests before the formula under consideration can be identified as probably consistent or probably inconsistent. However, it would seem that a good balance can only be struck with respect to the range of intended applications being addressed by the library.

6.3 Selection of good tests in a probable consistency system

Given a formula, and a probable consistency system, we may have a number of possible tests that we can apply. This raises the question of which tests are good tests to apply. Clearly there is a cost-benefit trade-off to consider. Shannon's information theory, as harnessed by Quinlan [Qui86], offers one way of formalizing the relative benefits of tests.

Suppose we start with $\alpha \in \mathcal{L}_r$, where $\mathcal{L}_r$ is the universe for a probable consistency system. Let the number of consistent formulae in $\mathcal{L}_r$ be p and the number of consistent formulae in $\mathcal{L}_r$ be n. Then the information (measured in bits) contained in the answer to whether α is consistent or not is given by the I function below.

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}$$

The result from a test function applied to α gives some information about whether α is consistent or not. Suppose, the test divides the set $\mathcal{L}_r$ into one of k sublanguages $\mathcal{L}_{r_1}, .., \mathcal{L}_{r_k}$ where each sublanguage $\mathcal{L}_{r_i}$ has

p_i consistent elements and n_i inconsistent elements. After undertaking test T, the remaining information that we require to classify whether $\alpha \in \mathcal{L}_r$ is consistent or inconsistent is given by the following function.

$$Remainder(T) = \sum_{i=1}^{k} \frac{p_i + n_i}{p + n} I(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i})$$

The information gain from test T is then defined as the following function.

$$Gain(T) = I(\frac{p}{p+n}, \frac{n}{p+n}) - Remainder(T)$$

So given a probable consistency system, the tests in it can be compared to determine which tests offer the better information gain. The net result of choosing a good combination of tests is that we can more efficiently determine whether or not there is a raised conditional probability of a formula α being inconsistent. To illustrate these definitions, consider the following example.

Example 6.1 *Suppose $\mathcal{L}_r$ is the following set of formulae:*

$$\begin{array}{ccc} \alpha \wedge \alpha & \alpha \wedge \beta & \beta \wedge \alpha \\ \alpha \wedge \neg\alpha & \alpha \wedge \neg\beta & \beta \wedge \neg\alpha \\ \neg\alpha \wedge \alpha & \neg\alpha \wedge \beta & \neg\beta \wedge \alpha \\ \neg\alpha \wedge \neg\alpha & \neg\alpha \wedge \neg\beta & \neg\beta \wedge \neg\alpha \end{array}$$

So $p = 10$ and $n = 2$. Since, $I(10/12, 2/12) = 0.65$ bits, we need 0.65 bits of information to determine whether a randomly selected member of $\mathcal{L}_r$ is consistent. Now suppose we consider a test T which determines whether a formula in $\mathcal{L}_r$ is composed from one propositional letter or two, then T splits $\mathcal{L}_r$ into $\mathcal{L}_1$ and $\mathcal{L}_2$, where $\mathcal{L}_1$ is

$$\alpha \wedge \alpha, \quad \alpha \wedge \neg\alpha, \quad \neg\alpha \wedge \alpha, \quad \neg\alpha \wedge \neg\alpha$$

and $\mathcal{L}_2$ is

$$\begin{array}{cccc} \alpha \wedge \beta & \beta \wedge \alpha & \alpha \wedge \neg\beta & \beta \wedge \neg\alpha \\ \neg\alpha \wedge \beta & \neg\beta \wedge \alpha & \neg\alpha \wedge \neg\beta & \neg\beta \wedge \neg\alpha \end{array}$$

So for $\mathcal{L}_1$, $p_1 = 2$ and $n_1 = 2$, and for $\mathcal{L}_2$, $p_2 = 8$ and $n_2 = 0$. Hence,

$$Gain(T) = 0.65 - \left[\frac{4}{12} I(\frac{2}{4}, \frac{2}{4}) + \frac{8}{12} I(\frac{8}{8}, \frac{0}{8})\right] = 0.32$$

Now consider an alternative test T' which determines whether a formula in $\mathcal{L}_r$ is composed from a negative literal or not, then T' splits $\mathcal{L}_r$ into $\mathcal{L}_3$ and $\mathcal{L}_4$, where $\mathcal{L}_3$ is

$$\alpha \wedge \alpha \quad \alpha \wedge \beta \quad \beta \wedge \alpha$$

and $\mathcal{L}_4$ is

$$\begin{array}{ccc} \alpha \wedge \neg\alpha & \alpha \wedge \neg\beta & \beta \wedge \neg\alpha \\ \neg\alpha \wedge \alpha & \neg\alpha \wedge \beta & \neg\beta \wedge \alpha \\ \neg\alpha \wedge \neg\alpha & \neg\alpha \wedge \neg\beta & \neg\beta \wedge \neg\alpha \end{array}$$

So for $\mathcal{L}_3$, $p_3 = 3$ and $n_3 = 0$, and for $\mathcal{L}_4$, $p_4 = 7$ and $n_4 = 2$. Hence,

$$Gain(T') = 0.65 - \left[\frac{3}{12} I(\frac{3}{3}, \frac{0}{3}) + \frac{9}{12} I(\frac{7}{9}, \frac{2}{9})\right] = 0.08$$

By comparing the gains for T and T', we see that T is a better test for $\mathcal{L}_r$ than T'.

It is possible to apply decision tree learning (e.g. ID3 [Qui86] or C4.5 [Qui93]) to select good tests from a space of possible tests. Indeed by evaluating the attributes of the examples, decision tree learning algorithms can define the good tests. In learning a decision tree, a table is constructed of training examples. Each row in the table corresponds to an individual example, and each column corresponds to an attribute. The final column is a classification of the example. The learning algorithm constructs a decision tree by selecting combinations of attributes for classifying these examples.

To apply decision tree learning to select good tests for a restricted language, and hence construct a probable consistency system, each row would correspond to a formula, the attributes would be the number of propositional letters used, the number of each type of logical symbols, the degree of nesting of each logical symbol, whether the formula is in conjunctive normal form, whether the form is a clause, etc, and the classification would be either consistent or inconsistent. Using the notion of gain, discussed above, a learning algorithm such as ID3 or C4.5 selects the "best" combination of tests for classifying each formula. Though in probable consistency checking we would not intend to apply all the tests in a branch of a decision tree because of the cost involved. Rather, enough tests to show that a formula is probably consistent or probably inconsistent (see Definition 3.9) would be undertaken.

Problems with using decision tree learning algorithms to construct a probable consistency system are: (1) Learning would need to be done over large numbers of examples for adequately large universes to be handled; and (2) some of the tests may be rather complex combinations of attributes, which may be relatively expensive to check. Advantages are: (1) the number of tests needed may be minimized; and (2) the process of determining the proportion of inconsistent formulae for each subset is automated by working in a bottom-up fashion, which is in contrast to the top-down approach of proving the proportion of inconsistent formulae in particular restricted languages (as illustrated in Section 4). In order to address the problem of creating large tables from which to learn the tests, it is possible to randomly generate some members of $\mathcal{L}_r$, and then use these as examples to learn good combinations of tests.

6.4 Use of non-uniform distributions over formulae

In using a uniform distribution over the likelihood of any formula being used in a particular restricted language (Definition 3.5), we are taking a syntactic approach. In theory, this is a reasonable assumption. It is the distribution with least commitment. In practice, we also believe it is a reasonable assumption if we know little or nothing about the underlying semantics (i.e. the meaning ascribed by the users) of the propositional letters in the language or of the way the language is used.

However, via experience, we may determine that some propositional letters are more likely to occur in the inconsistent formulae being used than others, or that the longer formulae being used are more likely to be inconsistent than the shorter formulae being used, or that certain symbols are more likely to be used in an inconsistent formula than others. Alternatively, we may have semantic, structural, or meta knowledge to indicate that the usage of the formulae by the user would not justify an assumption of a uniform distribution over the formulae. In any case, it may be appropriate to use a non-uniform distribution over the likelihood of each formula in each restricted language being used and a non-uniform distribution over the likelihood of a formula being inconsistent. We consider these issues in the following:

Frequency of usage of formulae in a restricted language If a formula is more likely to be used, then this can be reflected in a non-uniform distribution. For example, if we consider the language $\mathcal{L}_r$ that contains four formulae $\{\alpha_1, ..., \alpha_4\}$, where α_1 is inconsistent, and the remainder are consistent. Assuming a uniform distribution, the probability of inconsistency of any member of $\mathcal{L}_r$ being selected is $1/4$. However, suppose we also have data to suggest that α_1 and α_2 are each used twice as frequently as either of α_3 or α_4. Then, we may prefer a non-uniform distribution such that the probability of inconsistency of any member of $\mathcal{L}_r$ being selected is $1/3$.

Frequency of usage of inconsistent formulae in a restricted language If in a restricted language, a user is less likely to use an inconsistent formula than a consistent formula, then this can be reflected in a non-uniform distribution. For example, if we consider the language $\mathcal{L}_r$ that contains the 16 binary conjunctions that can be formed from the literals $\{\alpha, \neg\alpha, \beta, \neg\beta\}$. Assuming a uniform distribution, the probability of inconsistency of any member of $\mathcal{L}_r$ being selected is $1/4$. Since the formulae are quite simple, it might be five times more likely that a user selects a consistent formula than an inconsistent formula. If we take this into account, then we may prefer a non-uniform distribution such that the probability of inconsistency of any member of $\mathcal{L}_r$ being selected is $1/16$.

The frequency of usage of inconsistent formulae in a restricted language is actually a special case of the frequency of usage of formulae in a restricted language. Indeed all the factors that may influence the choice of a non-uniform distribution can be reduced to a set of constraints on the frequency of usage of formulae in each restricted language.

Machine learning offers a way of taking data about usage of formulae, and semantic or meta-level information about the formulae, to generate a non-uniform distribution. We may informally describe inductive machine learning as being a process of first generating a hypothesis (a general statement) from a set of training examples, and then checking the quality of the hypothesis with a set of test examples. If the hypothesis performs well with the test examples, then we may accept the hypothesis, otherwise we may amend or reject it.

The training and testing examples are taken from some class of all possible examples. So the training and testing are sensitive to the choice of examples selected. If the training examples are not representative, then the hypothesis will not be good with respect to the testing examples. In other words, the hypothesis will not be correct. The less representative the training examples are, the less correct the hypothesis will be. So to get good learning, the training examples should be representative of all the possible examples.

To investigate this issue, the notion of probably approximately correct (PAC) learning theory has been developed by Valiant [Val84]. This theory gives a a way of calculating the minimum number of training examples required to ensure that the training examples are representative of all possible examples assuming an upper limit to the degree of error ϵ for any good hypothesis and a maximum probability δ of a poor hypothesis being accepted by mistake. The error ϵ is used to define the notion of a hypothesis being approximately correct, and the probability δ is used to define the notion of a hypothesis being probably approximately correct.

If we learn non-uniform distributions over the likelihood of formulae being used in restricted languages, then we may have need of probably approximately correct (PAC) learning theory. Each of our hypotheses, in the form of a probability distribution represented by a set of conditional probability statements, is influenced by the choice or availability of representative data about the usage of the formulae in the restricted language. Therefore unrepresentative data about the usage of the formulae may lead to an inaccurate distribution and hence an inaccurate probable consistency system.

Note, if we use machine learning with the assumption of a uniform distribution, together with all the restricted language, to generate a hypothesis (i.e. the training set is the universe of all possible examples), as discussed in Section 6.3, then the hypothesis generated will be totally correct with respect to the class of all examples. Hence, the question of approximately correct and probably approximately correct learning has been circumvented, and so the use of PAC learning theory is not necessary. If we use machine learning with the assumption of a uniform distribution, together with radomly generated example a the restricted language, to generate a hypothesis, also discussed in Section 6.3, then the use of PAC learning may be necessary.

6.5 Probable consistency checking as an analytical framework

There is a rapidly growing literature on algorithms for consistency checking. One approach for propositional logic is to search for the existence of a satisfying model. These algorithms, called SAT algorithms, can be complete in the sense that they undertake an exhaustive search, and so try every possible model (e.g. the Davis-Puttnam procedure [DP60]), or they can be incomplete (e.g. the GSAT algorithm [SLM92]).

In SAT algorithms, the emphasis is on checking large sets of clauses, whereas probable consistency checking is focussed on individual formulae. Of course in classical logic, a conjunction of a set of formulae is a formula. So if larger formulae are considered in probable consistency checking, then the approaches coincide in that respect. However, the aim of SAT-algorithms is to just find an answer to the question of whether the set of formulae is consistent, whereas the aim of probable consistency checking is to analyse a formula, and to be able to explain the probability of consistency in terms of the results of the tests.

In practice, SAT algorithms and probable consistency can be complementary. Probable consistency checking can act as a filter of formulae that need SAT testing as a priority. Also probable consistency checking can evaluate formulae that have failed SAT testing. And probable consistency checking can highlight formulae that are potentially overconstrained, even if they are consistent, as discussed in Section 6.1.

The difference between the GSAT-type algorithms and probable consistency checking are further pronounced by the different types of uncertainty involved. In the GSAT-type algorithms, the uncertainty is whether the whole space of possible models has been considered. This uncertainty is clearly quite different to the probability of consistency of a formula conditioned on the membership of a particular restricted language.

However, via SAT algorithms, interesting observations have been made about the likelihood of consistency for certain types of satisfiability problem. In particular, there is now considerable empirical evidence linking the following ratio, which we will refer to as the α-ratio, for a set of clauses Δ, and the probability of consistency for Δ.

$$\frac{\text{The number of clauses in } \Delta}{\text{The number of different propositional letters used in the clauses in } \Delta}$$

The effect of the α-ratio on the probability of consistency has been studied for a range of languages, including binary and ternary clauses (see for example [MSL92]), and quantified Boolean formulae [Rin99]. A striking result of these studies is that the satisfiability problem displays phase transition behaviour [HHW96, MZK$^+$99]. When the α-ratio is low, the probability of Δ being inconsistent is very low, but as the α-ratio is increased, the probaility of inconsistency rises very sharply, until the probability of inconsistency reaches 1. This ratio can be used to define restricted languages, and the empirical results used to obtain conditional probability statements for a probable consistency system.

A further interesting observation with respect to the phase transition behaviour is that when the α-ratio is such that the probability of consistency is either very high or very low, then the cost of doing the satisfiability check is low. Whereas, when the α-ratio is such that the probability of consistency is around 0.5, then the cost of doing the satisfiability check is high. We can consider each of the cases as follows: (1) When the ratio is such that the probability of consistency is high, then there are many models that render the clauses satisfiable, and it is cheap to determine whether a model is satisfying; (2) When the ratio is such that the probability of consistency is low, then there are few models that render the clauses satisfiable, and it is cheap to reject the others; and (3) When the ratio is such that the probability of consistency is around 0.5, then there are generally a number of models that the render the clauses satisfiable, and that to reject each of the others involves a significant investigation.

A key problem in empirical approaches is differentiating easy problems from hard problems [MSL92, GW94]. When testing a new satisfiability algorithm on some set of formulae, how do we ensure that the test does not just involve formulae that are easy to check satifiability? Of course, we can take a comparative approach, using different algorithms on the same set of formulae. But that still seems unsatisfactory since our notion of easy and hard is relative to the algorithms that we use to explore the space of problems. The way around this problem is to seek ways of comparing sets of formulae that are external to the algorithms. In other words, we need analytical frameworks to complement the empirical frameworks. One such framework is built around the notion of phase transitions [HHW96, MZK$^+$99].

Another analytical approach is to use the notion of probable consistency systems. This contrasts with the empirical approaches in that the conditional probability statements are identified for relatively restricted languages, and that the emphasis is on the consistency of individial formulae that are relatively simple. In other words, the empirical approaches may be described as top-down whereas the probable consistency approach is bottom-up. This is reflected in the use of theories of statistical physics for analysing phase transition behaviour in the empirical studies. As an analogy, we may consider theoretical analyses based on phase transition behaviour as being akin to the use of fractals to model geomorhological features like coastlines, whereas analyses based on probable consistency checking are akin to the use of cartography to model particular geomorphological features, such as particular coastlines. The analyses are different but complementary. The former type of analysis provides a high-level view on consistency checking, highlighting key phenomena, whereas the later type of analyses provide a lower-level view that more closely reflects the details of the domain.

7 Discussion

Consistency checking is increasingly important in artificial intelligence, data and knowledge engineering, and software engineering [Hun96]. In approaches to knowledge representation and reasoning such as truth maintenance systems [Doy79, Kle86], argumentation systems based on identifying consistent subsets [MR70, Poo88, Bre89, BDP93, EGH95, BH01] and default reasoning systems [Rei80], consistency checking is an integral part of inferencing. Probable consistency checking potentially offers more efficient inferencing, either by directing conflict resolution to the more problematical areas of the data, or by allowing inferences before eliminating all possibilities of an inconsistency.

Inconsistency handling, and therefore consistency checking, is central to information fusion in logic [CH97]. This is the process of deriving a single consistent knowledgebase from multiple knowledgebases. The process is important in many cognitive tasks, such as decision-making, planning, design, specification, and regulation, that can involve collecting information, perhaps evidence, opinions, or preferences, from a number of potentially conflicting perspectives, or sources, or participants. Whilst theoretical approaches such as belief revision [AGM85, Gär88, DP98], databases and knowledgebase updating [FKUV86, KM89, Win90, Som94], and combining knowledgebases (for example [DLP92, BKMS91, BKMS92, Mot93]) are relevant, information fusion addresses a wider range of issues raised by practical imperatives in applications such as in developing specifications. These imperatives include the need to analyse inconsistent information, localize inconsistency, and to take context-dependent action in response to inconsistency. These needs make heavy use of consistency checking.

In a formal methods approach to developing specifications, a conflict can be manifested as a logical contradiction, giving rise to the need for consistency checking [FGH$^+$94, RNK98]. Elsewhere, we have considered the use of classical logic as a formalism for analysing conflicts in specifications [FGH$^+$94, HN97, HN98]. In developing specifications, inconsistencies at intermediate stages are not necessarily serious problems. Thus, missing inconsistencies may not have serious ramifications. This means that approximation techniques for consistency checking, where some inconsistencies may be missed, can be acceptable. Indeed, if it is the only general way of rendering consistency checking viable, then it is the most rational compromise.

Being tolerant of inconsistencies extends beyond developing specifications to a wider range of cognitive activities. Consider planning, designing, scheduling, and other kinds of problem solving in intelligent agents. Usually an intelligent agent has to compromise accuracy for efficiency. If a particular error occuring, manifested as an inconsistency, is not particularly important to an agent, then the effort expended by the agent in consistency checking is commensurately incomplete [GH91].

It seems that the real advantage with probable consistency checking in all these activities is the relative ordering we can obtain rather than the absolute probability values. The relative ordering helps prioritize search or further analysis. In effect, the use of probability theory in this approach gives a formal and reliable basis for heuristics for comparing and prioritizing a collection of sets of formulae.

So far we have restricted consideration of probable consistency checking to the propositional case of classical logic. However, we believe that the approach can be straightforwardly extended to the propositional forms of many modal and substructural logics. How we can extend probable consistency checking to the first-order case of classical logic is open.

Acknowledgements

I would like to thank James Cussens for some helpful feedback on this work.

References

[AGM85] C Alchourrón, P Gärdenfors, and D Makinson. On the logic of theory change: Partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50:513–530, 1985.

[BCP94] C Bourley, R Caferra, and N Peltier. A method for building models automatically: Experiments with an extension of OTTER. In *Proceedings of the Twelfth Conference on Computer-Aided Deduction (CADE'12)*, volume 814 of *Lecture Notes in Computer Science*, pages 72–86. Springer, 1994.

[BDP93] S Benferhat, D Dubois, and H Prade. Argumentative inference in uncertain and inconsistent knowledge bases. In *Proceedings of Uncertainty in Artificial Intelligence (UAI'93)*, pages 411–419. Morgan Kaufmann, 1993.

[BH01] Ph Besnard and A Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128:203–235, 2001.

[BKMS91] C Baral, S Kraus, J Minker, and V Subrahmanian. Combining multiple knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 3(2), 1991.

[BKMS92] C Baral, S Kraus, J Minker, and V Subrahmanian. Combining knowledgebases of first-order theories. *Computational Intelligence*, 8:45–71, 1992.

[Bre89] G Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *Proceedings of the Eleventh International Conference on Artificial Intelligence (IJCAI'89)*, pages 1043–1048, 1989.

[CH97] L Cholvy and A Hunter. Information fusion in logic: A brief overview. In D Gabbay, R Kruse, A Nonnengart, and H-J Ohlbach, editors, *Qualitative and Quantitative Practical Reasoning*, volume 1244 of *Lecture Notes in Artificial Intelligence*, pages 86–95. Springer, 1997.

[CP95] R Caferra and N Peltier. Model building and interactive theory discovery. In *Fourth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, volume 918 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 1995.

[CZ93] R Caferra and N Zabel. Building models by using tableaux extended by equational problems. *Journal of Logic and Computation*, 3:3–25, 1993.

[Dar99] A Darwiche. Compiling knowledge into decomposible negation normal form. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 284–289, 1999.

[DG94] W Dowling and H Gallier. Linear time algorithms for testing the satisfiability of propositional horn formula. *Journal of Logic Programming*, 3:267–284, 1994.

[DLP92] D Dubois, J Lang, and H Prade. Dealing with multi-source information in possibilistic logic. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 38–42, 1992.

[Doy79] J Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[DP60] M Davis and H Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[DP87] R Dechter and J Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[DP98] D Dubois and H Prade. *Handbook of Defeasible Reasoning and Uncertainty Management*, volume 1. Kluwer, 1998.

[EGH95] M Elvang-Gøransson and A Hunter. Argumentative logics: Reasoning from classically inconsistent information. *Data and Knowledge Engineering*, 16:125–145, 1995.

[FGH+94] A Finkelstein, D Gabbay, A Hunter, J Kramer, and B Nuseibeh. Inconsistency handling in multi-perspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994.

[FKUV86] R Fagin, G Kuper, J Ullman, and M Vardi. Updating logical databases. *Advances in Computing Research*, 3:1–18, 1986.

[Gär88] P Gärdenfors. *Knowledge in Flux: Modelling the Dynamics of Epistemic States*. MIT Press, 1988.

[GH91] 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.

[GJ79] M Garey and D Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[GM94] M Ginsberg and D McAllester. GSAT and dynamic backtracking. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR'94)*, pages 226–237, 1994.

[GW93] I Gent and T Walsh. Towards an understanding of hill climbing procedures for SAT. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, pages 28–33, 1993.

[GW94] I Gent and T Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70:335–345, 1994.

[HHW96] T Hogg, B Huberman, and C Williams. Phase transitions and the search problem. *Artificial Intelligence*, 81:1–15, 1996.

[HN97] A Hunter and B Nuseibeh. Analysing inconsistent specifications. In *Proceedings of 3rd International Symposium on Requirements Engineering (ISRE'97)*, pages 78–86. IEEE Computer Society Press, 1997.

[HN98] A Hunter and B Nuseibeh. Managing inconsistent specifications: Reasoning, analysis and action. *ACM Transactions in Software Engineering and Methodology*, 7:335–367, 1998.

[Hun96] A Hunter. *Uncertainty in Information Systems*. McGraw-Hill, 1996.

[Hun00a] A Hunter. Merging potentially inconsistent items of structured text. *Data and Knowledge Engineering*, 34:305–332, 2000.

[Hun00b] A Hunter. Ramification analysis using causal mapping. *Data and Knowledge Engineering*, 32:1–27, 2000.

[Hun01] A Hunter. Hybrid argumentation systems for structured news reports. *Knowledge Engineering Review*, 2001, 16:295–329.

[Hun02a] A Hunter. Logical fusion rules for merging structured news reports. *Data and Knowledge Engineering*, 42:23–56, 2002.

[Hun02b] A Hunter. Measuring inconsistency in knowledge via quasi-classical models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'02).* pages 68–73, MIT Press, 2002.

[Hun02c] A Hunter. Merging structured text using temporal knowledge. *Data and Knowledge Engineering*, 41:29–66, 2002.

[Kle86] J De Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[KM89] H Katsuno and A Mendelzon. A unified view of propositional knowledgebase updates. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, 1989.

[Lev84] H Levesque. A logic of implicit and explicit belief. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'84)*, pages 198–202, 1984.

[Mar95] P Marquis. Knowledge compilation using prime implicates. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 837–843, 1995.

[Mot93] A Motro. A formal framework for integrating inconsistent answers from multiple information sources. Technical Report ISSE-TR-93-106, George Mason University, 1993.

[MR70] R Manor and N Rescher. On inferences from inconsistent information. *Theory and Decision*, 1:179–219, 1970.

[MSG97] B Mazure, L Sais, and E Gregoire. Checking several forms of consistency in non-monotonic knowledgebases. In D Gabbay, R Kruse, A Nonnengart, and H-J Ohlbach, editors, *Qualitative and Quantitative Practical Reasoning*, volume 1244 of *Lecture Notes in Artificial Intelligence*, pages 123–130. Springer, 1997.

[MSL92] D Mitchell, B Selman, and H Levesque. Hard and easy distributions of sat problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 459–464, 1992.

[MZK+99] R Monasson, R Zecchina, S Kirkpatrick, B Selman, and L Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, 1999.

[OS88] F Oppacher and E Suen. HARP: A tableau-based theorem prover. *Journal of Automated Reasoning*, 4:69–100, 1988.

[Pap94] C Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Poo88] D Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[Qui86] J Quinlan. Induction of decision trees. *Machine Learning*, 1:86–106, 1986.

[Qui93] J Quinlan. *C4.5: Programs for Learning*. Morgan Kaufmann, 1993.

[Rei80] R Reiter. Default logic. *Artificial Intelligence*, 13:81–132, 1980.

[Rin99] J Rintanen. Improvements to the evaluation of quantified boolean formulae. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 1192–1197. Morgan Kanfmann, 1999.

[RNK98] A Russo, B Nuseibeh, and J Kramer. Restructuring requirements specifications for managing inconsistency and change: A case study. In *Proceedings of the Third IEEE International Conference on Requirements Engineering (ICRE'98)*, 1998.

[SC95] M Schaerf and M Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74:249–310, 1995.

[Sch94] J Schumann. SETHEO V3.2: Recent developments. In *Proceedings of the Twelfth Conference on Computer-Aided Deduction (CADE'12)*, volume 814 of *Lecture Notes in Computer Science*. Springer, 1994.

[Sla96] J Slaney. FINDER: Finite domain enumerator version 3 notes and guide. Technical report, Centre for Information Science Research, Australian National University, 1996.

[SLM92] 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.

[Som94] Leá Sombé. *Revision and Updating in Knowledgebases*. Wiley, 1994.

[Val84] L Valiant. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27:1134–1142, 1984.

[Win90] M Winslett. *Updating Logical Databases*. Cambridge University Press, 1990.