ARTICLE IN PRESS

International Journal of Approximate Reasoning xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

International Journal of Approximate Reasoning

journal homepage: www.elsevier.com/locate/ijar

From inconsistency handling to non-canonical requirements management: A logical perspective

Kedian Mu[a,*], Jun Hong[b], Zhi Jin[c], Weiru Liu[b]

[a] *School of Mathematical Sciences, Peking University, Beijing 100871, PR China*
[b] *School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, BT7 1NN, UK*
[c] *Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, PR China*

ARTICLE INFO

Article history:
Received 21 July 2011
Revised 2 May 2012
Accepted 25 July 2012

Available online xxx

Keywords:
Inconsistency handling
Non-canonical requirements
Redundancy
Incompleteness
Requirements engineering

ABSTRACT

As a class of defects in software requirements specification, inconsistency has been widely studied in both requirements engineering and software engineering. It has been increasingly recognized that maintaining consistency alone often results in some other types of non-canonical requirements, including incompleteness of a requirements specification, vague requirements statements, and redundant requirements statements. It is therefore desirable for inconsistency handling to take into account the related non-canonical requirements in requirements engineering. To address this issue, we propose an intuitive generalization of logical techniques for handling inconsistency to those that are suitable for managing non-canonical requirements, which deals with incompleteness and redundancy, in addition to inconsistency. We first argue that measuring non-canonical requirements plays a crucial role in handling them effectively. We then present a measure-driven logic framework for managing non-canonical requirements. The framework consists of five main parts, identifying non-canonical requirements, measuring them, generating candidate proposals for handling them, choosing commonly acceptable proposals, and revising them according to the chosen proposals. This generalization can be considered as an attempt to handle non-canonical requirements along with logic-based inconsistency handling in requirements engineering.

© 2012 Elsevier Inc. All rights reserved.

1. Introduction

A good-quality software requirements specification is crucial for the success of a software development project. However, it can often be difficult to develop such a requirements specification. Problems associated with requirements are still potentially the major causes of software project failures, and there is a great need to facilitate software development process [1].

Inconsistency has been considered as a main class of defects in requirements specifications [1]. A number of techniques have been proposed to handle inconsistency in requirements engineering [2–12]. In particular, it has increasingly been recognized that it is effective to use logics to formulate management of inconsistent requirements specifications [1]. Various logic-based approaches to handling inconsistency in requirements specifications have recently been proposed [2,4,1,9–12]. Most of these approaches focus on how to apply non-classical reasoning techniques, such as paraconsistent reasoning and non-monotonic reasoning, to detecting and analyzing inconsistency in requirements specifications. For example, Hunter and Nuseibeh [2] developed labeled quasi-classic logic to represent and reason about requirements specifications in the presence of inconsistency. Gervasi and Zowghi [1] proposed methods for reasoning about inconsistency in natural language

[*] Corresponding author.
E-mail address: mukedian@math.pku.edu.cn

0888-613X/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ijar.2012.07.006

requirements by combining natural language parsing techniques and non-monotonic reasoning. Easterbrook and Chechik [4] presented a framework termed χbel for merging inconsistent viewpoints using multi-valued logics.

However, inconsistency is not an isolated problem in requirements engineering. The process of maintaining consistency often results in some other undesirable types of information about requirements. For example, Zowghi and Gervasi [12] argued that there is an important causal relationship between consistency, completeness and correctness of requirements in requirements evolution. Recently, Martinez et al. [11] took into account the interplay between inconsistency and incompleteness in merging multi-perspective requirements. Previously, we termed requirements that are incomplete, redundant, vague or inconsistent as non-canonical requirements [13].

Intuitively, a feasible proposal for inconsistency handling should take into account other non-canonical requirements that are also involved in inconsistency. Previously, we have developed some logical tools for managing other non-canonical requirements as well as inconsistency. For example, we have presented a logical tool for uniformly characterizing inconsistency, incompleteness, vagueness, and redundancy in requirements specifications [13]. We have also proposed an approach to detecting inconsistency, incompleteness and redundancy in requirements specifications based on Answer Set Programming [14]. These methods focus on how to identify non-canonical requirements when detecting inconsistency rather than on the problem of systematically managing non-canonical requirements.

It has increasingly been recognized that measuring inconsistency is crucial for effectively managing inconsistency in requirements [5,15,11]. Furthermore, appropriate measurements of non-canonical requirements could provide a good basis for making some trade-off decisions on handling non-canonical requirements. For example, software developers would need to know whether a requirements specification would become more incomplete if some inconsistent requirements are removed. They would also need to know whether some requirements changes for resolving inconsistency in a requirements specification would increase the degree of redundancy in the specification. On the other hand, the most redundant requirements involved in inconsistency should be given priority for consideration when there are options for requirements changes.

To address these issues, in this paper, we propose a logical framework for managing non-canonical requirements, including inconsistent, incomplete, and redundant requirements. First, we formulate a requirements specification as a set of logical formulas. Second, we present a framework for managing non-canonical requirements, which consists of five main parts, identifying non-canonical requirements, measuring them, generating candidate proposals for handling them, choosing commonly acceptable proposals, and revising them according to the chosen proposals. Third, we provide a family of measures for non-canonical requirements, including measures for inconsistency, incompleteness, and redundancy. Finally, we discuss some strategies for generating proposals for handling non-canonical requirements by using these measures.

The rest of this paper is organized as follows. Section 2 gives a brief introduction to the logical representation of requirements. We propose a general framework for managing non-canonical requirements in Section 3. Section 4 proposes measures for non-canonical requirements. Section 5 discusses strategies for generating proposals for handling non-canonical requirements. We compare our approach with related work in Section 6. We conclude the paper in Section 7.

2. Preliminaries

First-order logic has increasingly been considered as a promising tool for representing requirements [2,9,10]. Moreover, restricting first-order logic to propositional logic is a useful and practical way of balancing the computational advantages of propositional logic and its limited expressive power in requirements engineering as well as software engineering [1,9,10,16]. In this paper, we use a classical first-order language without function symbols and existential quantifiers. Classical first-order logic is the most convenient to illustrate our approach, as will be seen in the rest of the paper.

Let $\mathscr{P}$ be a set of predicate symbols, $\mathscr{V}$ be a set of variable symbols, and $\mathscr{C}$ a set of constant symbols. We call $\mathscr{A} = \{p(q_1, \ldots, q_n) | p \in \mathscr{P}$ and $q_1, \ldots, q_n \in \mathscr{V} \cup \mathscr{C}\}$ the set of atoms. Let $\mathscr{F}$ be the set of classical formulas formed from a set of atoms $\mathscr{A}$ and a set of logical connectives $\{\vee, \wedge, \neg, \rightarrow\}$.

In particular, we call $p(q_1, \ldots, q_n)$ a ground atom if and only if $q_1, \ldots, q_n$ are all constant symbols. Let $\mathscr{A}_0$ be a set of ground atoms and $\mathscr{F}_0$ be a set of classical formulas formed from a set of ground atoms $\mathscr{A}_0$ and a set of logical connectives $\{\vee, \wedge, \neg, \rightarrow\}$.

Let $\mathscr{G}$ be the set of ground formulas and prenex universally quantified formulas formed from $\mathscr{F}$, where $\forall X_1 \cdots \forall X_n \alpha \in \mathscr{G}$ if $\alpha \in \mathscr{F}$, and $X_1, \ldots, X_n$ are the free variables in α.

We can use formulas in $\mathscr{G}$ to formulate requirements expressed in natural language. For example, we can represent a requirement,"*If an authorized user requests to borrow a book and the book is available, then the user can borrow the book*", as

$$\forall \texttt{User} \forall \texttt{Book}\ (\texttt{auth}(\texttt{User}) \wedge \texttt{requ}(\texttt{User}, \texttt{Book}) \wedge \texttt{avai}(\texttt{Book}) \rightarrow \texttt{borr}(\texttt{User}, \texttt{Book})).$$

A scenario is a potential application setting of the system-to-be. For the sake of simplicity, we assume that a scenario consists of two parts, a set of facts to model the scenario, and a set of expected responses to model the expected behavior of the system-to-be. Moreover, facts and expected responses can be formulated by ground formulas in $\mathscr{F}_0$. We use $\langle S_I, S_E \rangle$ to denote a scenario, where S_I and S_E are the set of facts and the set of expected responses, respectively. Intuitively, we consider that scenarios that correctly describe the application domains are consistent. When there is no confusion, we abbreviate a scenario $\langle S_I, S_E \rangle$ as S. Suppose that a scenario S for the above requirement contains the following facts:

- *Alice is an authorized user*;
- *Alice requests to borrow the Software Engineering book*;
- *A Software Engineering book* is available.

and the following expected response:

- *Alice can borrow the Software Engineering book*.

We can then represent the scenario as $\langle S_I, S_E \rangle$, where

$$S_I = \{\texttt{auth}(\texttt{Alice}),\ \texttt{requ}(\texttt{Alice}, \texttt{Soft\_eng}),\ \texttt{avai}(\texttt{Soft\_eng})\},$$
$$S_E = \{\texttt{borr}(\texttt{Alice}, \texttt{Soft\_eng})\}.$$

However, to check inconsistency in requirements specifications, universally quantified formulas are always instantiated with constants in the given scenarios. For example, given the above scenario S, we use the following ground formula instead of the universally quantified formula above:

$$\texttt{auth}(\texttt{Alice}) \wedge \texttt{requ}(\texttt{Alice}, \texttt{Soft\_eng}) \wedge \texttt{avai}(\texttt{Soft\_eng}) \rightarrow \texttt{borr}(\texttt{Alice}, \texttt{Soft\_eng}).$$

Generally, if ground formulas $\alpha_1, \alpha_2, \ldots, \alpha_n$ are the instantiations of the universally quantified formula α with different facts in a scenario, we may use $\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$ instead of α in the scenario. Thus, we concentrate on the instantiated requirements in the rest of this paper. That is, we assume that a set of requirements with regard to a given scenario can be formulated by a set of ground formulas in $\mathscr{F}_0$. With this in mind, we can restrict the first-order logical representation of requirements to the propositional case in reasoning about requirements.

We call a finite set of formulas in $\mathscr{G}$ a knowledge base. In particular, we call a knowledge base a requirements specification if each formula in the knowledge base represents a requirement. As pointed out in [17], the formulation of requirements using formulas should be syntax sensitive, that is, any two knowledge bases that contain two different sets of formulas are considered as two distinct requirements specifications even if the two knowledge bases are logically equivalent. For example, $\{\alpha, \beta\}$ represents a set of two requirements α and β, but $\{\alpha \wedge \beta\}$ means that there is only one requirement, $\alpha \wedge \beta$ [17]. Bear this in mind,when there is no confusion we make no distinction between a knowledge base and a requirements specification in the rest of this paper.

Let $\vdash$ be the classical consequence relation, then $K \vdash K'$ if $K \vdash \alpha$ for all $\alpha \in K'$.

A knowledge base K is inconsistent if there is a formula α such that $K \vdash \alpha$ and $K \vdash \neg\alpha$. We abbreviate $\alpha \wedge \neg\alpha$ as $\perp$ when there is no confusion. Thus an inconsistent knowledge base K is denoted by $K \vdash \perp$.

Previously, we have defined inconsistency, incompleteness and redundancy in requirements with regard to a given scenario, respectively.

Definition 2.1 (*Inconsistency* [13]). Let $S = \langle S_I, S_E \rangle$ be a scenario and R a requirements specification for S, then R is inconsistent with regard to S if $R \cup S_I \vdash \perp$.

Definition 2.2 (*Incompleteness* [13]). Let $S = \langle S_I, S_E \rangle$ be a scenario and R a requirements specification for S, then R is incomplete with regard to S if $\exists \alpha \in S_E$ such that $R \cup S_I \nvdash \alpha$.

Definition 2.3 (*Redundancy* [13]). Let $S = \langle S_I, S_E \rangle$ be a scenario and R a requirements specification for S, then R is redundant with regard to S if $\exists R' \subset R\ s.t.\ R' \cup S_I \nvdash \perp$ and $R' \cup S_I \vdash S_E$. Furthermore, we call R' a simplification of R w.r.t. S.

Note that the incompleteness of R w.r.t. S means that R does not contain requirements about some expected behaviors of the system-to-be in S. In contrast, the redundancy of R w.r.t. S implies that R conveys redundant information (consistent or not) about expected behaviors of the system-to-be in R.

We use the following example to illustrate the formulation of requirements in the form of classical logic formulas.

Example 2.1. Consider the following requirements specification of a residential estate management system, which manages the vehicle entrance of the estate.

(r1) Vehicles without authorization by the Management Board of the residential estate are not allowed to enter the estate.

(r2) Vehicles with authorization by the Management Board of the residential estate can enter the estate.
(r3) The system triggers the warning alarm if a vehicle without authorization enters the estate.
(r4) If the system triggers the warning alarm, the driver of the vehicle cannot activate the button for entrance.
(r5) An emergency vehicle can enter the estate, and no authorization by the Management Board of the residential estate is required.
(r6) Vehicles without authorization except emergency ones cannot enter the estate.
(r7) Drivers ofvehicles without authorization except emergency ones cannot activate the button for entrance.

First, we translate these requirements into logic formulas. Suppose that we use

- predicate $\texttt{Aut}(\texttt{x})$ to denote that $\texttt{x}$ is authorized by the Management Board of the residential estate, where $\texttt{x}$ is a vehicle;
- predicate $\texttt{Ent}(\texttt{x})$ to denote that $\texttt{x}$ enters the residential estate;
- predicate $\texttt{Ala}(\texttt{x})$ to denote that the system triggers the warning alarm if $\texttt{x}$ enters the estate;
- predicate $\texttt{Act}(\texttt{x}, \texttt{y})$ to denote that $\texttt{x}$'s driver activates button $\texttt{y}$;
- predicate $\texttt{Eme}(\texttt{x})$ to denote that $\texttt{x}$ is an emergency vehicle;
- constant $\texttt{entr}$ to denote the button for entrance.

Each requirement statement can be represented by a formula:

(r1) $(\forall \texttt{x})(\neg\texttt{Aut}(\texttt{x}) \rightarrow \neg\texttt{Ent}(\texttt{x}))$,
(r2) $(\forall \texttt{x})(\texttt{Aut}(\texttt{x}) \rightarrow \texttt{Ent}(\texttt{x}))$,
(r3) $(\forall \texttt{x})(\neg\texttt{Aut}(\texttt{x}) \rightarrow \texttt{Ala}(\texttt{x}))$,
(r4) $(\forall \texttt{x})(\texttt{Ala}(\texttt{x}) \rightarrow \neg\texttt{Act}(\texttt{x}, \texttt{entr}))$,
(r5) $(\forall \texttt{x})(\texttt{Eme}(\texttt{x}) \rightarrow \texttt{Ent}(\texttt{x}) \wedge \neg\texttt{Aut}(\texttt{x}))$,
(r6) $(\forall \texttt{x})(\neg\texttt{Aut}(\texttt{x}) \wedge \neg\texttt{Eme}(\texttt{x}) \rightarrow \neg\texttt{Ent}(\texttt{x}))$,
(r7) $(\forall \texttt{x})(\neg\texttt{Aut}(\texttt{x}) \wedge \neg\texttt{Eme}(\texttt{x}) \rightarrow \neg\texttt{Act}(\texttt{x}, \texttt{entr}))$.

We can then use the above set of formulas to represent the given requirements specification as:

$$R = \{(\text{r1}), (\text{r2}), (\text{r3}), (\text{r4}), (\text{r5}), (\text{r6}), (\text{r7})\}.$$

Second, we verify the preliminary requirements based on some scenarios. Now we consider the first scenario S_1 about fire engines. A fire engine should be considered as an emergence vehicle. Suppose that we use constant $\texttt{fi\_en}$ to denote a fire engine, then $S_1 = \langle S_{1_I}, S_{1_E}\rangle$ may be given as follows:

$$S_{1_I} = \{\texttt{Eme}(\texttt{fi\_en})\}, \quad S_{1_E} = \{\texttt{Ent}(\texttt{fi\_en})\}.$$

Clearly,

$$R \cup S_{1_I} \vdash \texttt{Ent}(\texttt{fi\_en}) \wedge \neg\texttt{Ent}(\texttt{fi\_en}).$$

That is, R is inconsistent w.r.t. S_1.

Consider the second scenario S_2 about other vehicles. Suppose that $\texttt{vis1}$ is a visitor's car with authorization, and $\texttt{vis2}$ is a visitor's car without authorization, then $S_2 = \langle S_{2_I}, S_{2_E}\rangle$ may be given as follows:

$$S_{2_I} = \{\texttt{Aut}(\texttt{vis1}), \neg\texttt{Aut}(\texttt{vis2}), \neg\texttt{Eme}(\texttt{vis1}), \neg\texttt{Eme}(\texttt{vis2})\},$$
$$S_{2_E} = \{\texttt{Ent}(\texttt{vis1}), \neg\texttt{Ent}(\texttt{vis2})\}.$$

Clearly, R is redundant w.r.t. S_2, since

$$R \setminus \{\neg\texttt{Aut}(\texttt{x}) \rightarrow \neg\texttt{Ent}(\texttt{x})\} \cup S_{2_I} \vdash S_{2_E}.$$

Consider the third scenario S_3 about the alarm. Suppose that $\texttt{vis3}$ is a visitor's car without authorization. In this scenario, we expect that its driver can activate the button for help when the warning alarm has been triggered, then $S_3 = \langle S_{3_I}, S_{3_E}\rangle$ may be given as follows:

$$S_{3_I} = \{\neg\texttt{Aut}(\texttt{vis3}), \neg\texttt{Eme}(\texttt{vis3})\}, \quad S_{3_E} = \{\neg\texttt{Ent}(\texttt{vis3}), \texttt{Act}(\texttt{vis3}, \texttt{help})\}.$$

Clearly, R is incomplete w.r.t. S_3, since

$$R \cup S_{3_I} \nvdash \texttt{Act}(\texttt{vis3}, \texttt{help}).$$

3. A logical framework for managing non-canonical requirements

Roughly speaking, approaches for effectively managing non-canonical requirements should focus on revising non-canonical requirements in addition to detecting them. However, techniques for identifying appropriate actions for revising non-canonical requirements are still to be developed. Even if we consider inconsistency handling only in requirements engineering, it is difficult to provide a common methodology for handling all inconsistencies in requirements, since inconsistency handling actions are rather context-sensitive [2,18,19]. This means that a practical approach to revising non-canonical requirements should focus on how to identify requirements to be changed rather than on how to change them.

It has increasingly been recognized that appropriate measures for inconsistent requirements provide a good basis for making necessary trade-off decisions on resolving inconsistencies [5,11,15]. However, as mentioned earlier, inconsistency handling in requirements is not an isolated problem, which is often associated with incompleteness and redundancy. It is necessary to take into account measurements for incompleteness and redundancy when handling inconsistency based on the measurement for inconsistency. To address this issue, we propose a measure-driven framework for managing non-canonical requirements, as shown in Fig. 1. Central to this framework is the issue on measuring non-canonical requirements, which provides a basis for identifying appropriate proposals for resolving these non-canonical requirements. It should be noted that this framework supports an iterative and evolutionary process for achieving a good software requirements specification. The framework can be used when requirements analysts validate a software requirements specification with given scenarios, or when some of the existing requirements in the specification have been changed.

The framework takes as input a software requirements specification to be validated with some given scenarios. Its output is a consistent and complete simplification of the requirements specification with regard to given scenarios. The framework consists of five activities for managing non-canonical requirements:

- *Identifying non-canonical requirements*. This activity is concentrated on detecting non-canonical requirements with regard to the given scenarios. Most of the current logical approaches to identifying non-canonical requirements are

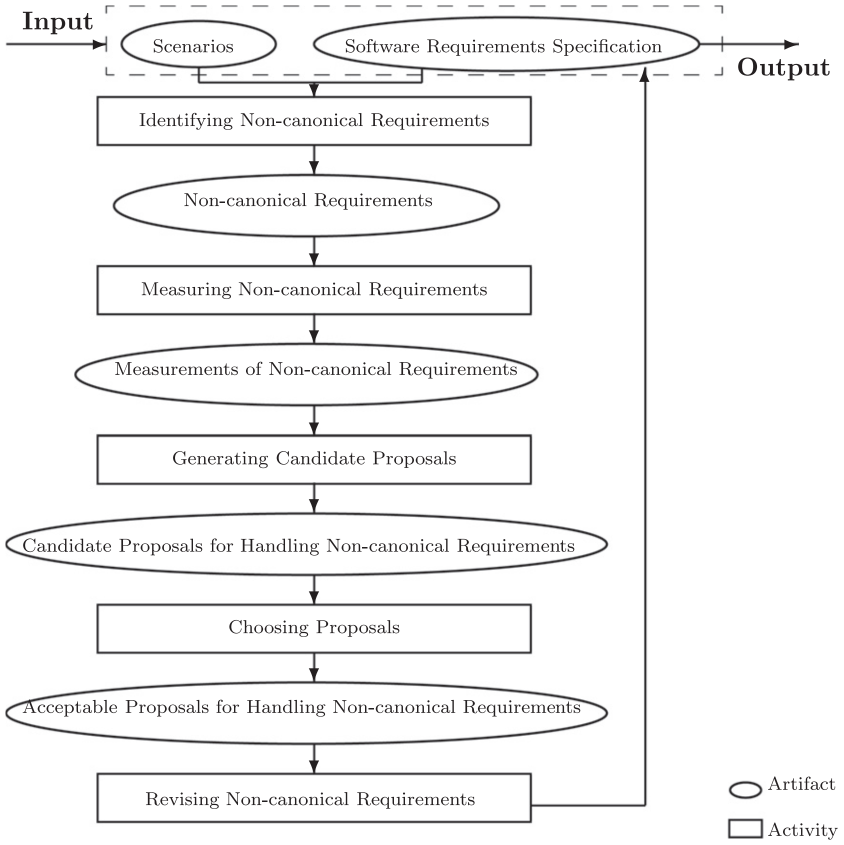
Input
Scenarios
Software Requirements Specification
Output
Identifying Non-canonical Requirements
Non-canonical Requirements
Measuring Non-canonical Requirements
Measurements of Non-canonical Requirements
Generating Candidate Proposals
Candidate Proposals for Handling Non-canonical Requirements
Choosing Proposals
Acceptable Proposals for Handling Non-canonical Requirements
Revising Non-canonical Requirements
Artifact
Activity

Fig. 1. A Framework for Managing Non-canonical Requirements focused only on inconsistency checking by reasoning with requirements. In our previous paper [14], we proposed a logical approach to identifying non-canonical requirements using the Answer Set Programming technique. We argued that the declarative nature of Answer Set Programming and its support for non-monotonicity make it suitable for formulating the problem of identifying non-canonical requirements [14]. Informally speaking, given a scenario, the problem of finding the behaviors of the system-to-be in the scenario can be reformulated into a problem of Answer Set Programming, which considers the expected response of the scenario as an intended answer. By comparing the answers computed by Answer Set Programming and the intended answer, inconsistency, incompleteness, and redundancy can be identified using appropriate strategies [14].

- *Measuring non-canonical requirements*. It has increasingly been recognized that an appropriate measure for inconsistent requirements may provide a good basis for making necessary trade-off decisions. Some approaches to measuring inconsistent requirements have been proposed [15,20]. In comparison, relatively few approaches to measuring incomplete and redundant requirements have been proposed. However, it is potentially important to measure non-canonical requirements in order to handle them. For example, developers would need to know how redundant a requirements specification is, in particular, the blame on each requirement for the redundancy in the requirements specification, before they can make a decision on how to simplify it. Developers would also often need to monitor whether the revised specification is less redundant (resp. less incomplete or less inconsistent) than the original one. In the next section, we propose a family of measures for non-canonical requirements.
- *Generating candidate proposals for handling non-canonical requirements*. When non-canonical requirements are identified, appropriate actions for revising them would need to be proposed. However, it has been widely recognized that proposing appropriate actions for handling non-canonical requirements, such as inconsistent requirements, is an important challenge in requirements engineering, since handling non-canonical requirements is a rather context-sensitive process, which always involves a process of interaction and competition among the corresponding stakeholders [9]. It is difficult to provide a common methodology for revising all the non-canonical requirements in requirements engineering. Consequently, a feasible general framework for handling non-canonical requirements should focus on identifying requirements that should be changed rather than how these requirements should be revised. Intuitively, identifying requirements that are involved in inconsistency (resp. redundancy) in a requirements specification should be changed depends on measuring non-canonical requirements, especially, on measuring the blame on each requirement involved. We will provide an approach to generating candidate proposals for handling non-canonical requirements later, which is driven by the measures of non-canonical requirements.
- *Choosing commonly acceptable proposals for handling non-canonical requirements*. This activity is designed to identify acceptable proposals for handling non-canonical requirements from candidate proposals, based on preference relations over the candidate proposals given by the stakeholders involved in non-canonical requirements. In requirements engineering practice, the stakeholders may provide different proposals for handling non-canonical requirements from their own perspective. Each stakeholder also assesses different proposals from her/his own perspective and gives preferences on them according to her/his expectations. Developers would then need to identify proposals for handling non-canonical requirements that are commonly acceptable using appropriate group decision making mechanisms, such as negotiation [9] and combinatorial voting [21].
- *Revising non-canonical requirements according to commonly acceptable proposals*. This activity is designed to revise the original requirements specification into a well structured requirements specification. It represents the action of revising the original requirements specification based on the selected proposals by stakeholders. Note that the revision according to the commonly acceptable proposals may lead to subsequent iteration of this process.

Note that we have only described the main activities in this process, some other activities for enhancing communication with stakeholders, such as translating requirements into formal logic formulas and translating logic-formulas-based proposals into natural language sentences are omitted.

4. Measuring non-canonical requirements

4.1. Measuring inconsistent requirements

We focus on two kinds of measures for inconsistent requirements, i.e., the measure for inconsistency in a whole set of requirements and the measure for the blame on each requirement for inconsistency in a set of requirements. An increasing number of measures for inconsistency in a whole knowledge base have been proposed in many applications. In contrast, there are relatively few measures for the blame on each formula for inconsistency in a knowledge base. The Shapley inconsistency value presented in [22] is one of the most representative measures for the blame on each formula for inconsistency in a knowledge base, which allows a given assessment of inconsistency for a whole knowledge base to be distributed to each formula by using a coalition game model – Shapley value. In particular, Hunter and Konieczny [17] argued that it is natural to measure inconsistency in a knowledge base through minimal inconsistent subsets if we consider the minimal inconsistent subsets of a knowledge base as the purest form of inconsistency in the knowledge base, and they proposed the MI inconsistency measure (i.e., the number of minimal inconsistent subsets) to measure the inconsistency in a knowledge base.

Furthermore, they proposed the MinInc inconsistency value, a special Shapley inconsistency value based on only minimal inconsistent subsets of a knowledge base, to measure the blame on each formula for inconsistency in the knowledge base [17]. Previously, we revised these measures and presented weighted MI inconsistency measure and weighted MinInc inconsistency value to assess the degree of inconsistency in a knowledge base and the blame on each formula for inconsistency in the knowledge base, respectively [20].

An inconsistent knowledge base K is called a *minimal inconsistent set* (or *minimal inconsistent knowledge base*) if none of its proper subsets is inconsistent. If $K' \subseteq K$ and K' is a minimal inconsistent set, then we call K' a *minimal inconsistent subset* of K. We use $\mathsf{MI}(K)$ to denote the set of minimal inconsistent subsets of K, i.e.,

$$\mathsf{MI}(K) = \{K' \subseteq K | K' \vdash \bot \text{ and for all } K'' \subset K', K'' \nvdash \bot\}.$$

Minimal inconsistent subsets can be considered as the purest form of inconsistency for syntax sensitive conflict resolution, since one only needs to remove one formula from each minimal inconsistent subset in such cases [23]. From this perspective, minimal inconsistent subsets have been considered as one of the intuitive constructions for developing an inconsistency measure [17].

A formula in K is called a *free formula* if it does not belong to any minimal inconsistent subset of K [22]. That is, the free formulas in K have nothing to do with the minimal inconsistent subsets of K. We use $\mathsf{FREE}(K)$ to denote the set of free formulas in K, i.e.,

$$\mathsf{FREE}(K) = \{\alpha \in K | \text{ for all } M \in \mathsf{MI}(K), \alpha \notin M\}.$$

Definition 4.1 (*Weighted MI inconsistency measure* [20]). Let K be a knowledge base. Then the weighted MI inconsistency measure for K, denoted as $\mathsf{I}_\mathsf{W}(K)$, is defined as follows [1]:

$$\mathsf{I}_\mathsf{W}(K) = \sum_{M \in \mathsf{MI}(K)} \frac{1}{|M|}.$$

Essentially, the weighted MI inconsistency measure for a knowledge base K is the sum of inconsistency measures for all the minimal inconsistent subsets of K, while the inconsistency in a minimal inconsistent subset M is captured by $\frac{1}{|M|}$. Note that $\mathsf{I}_\mathsf{W}(M) = \frac{1}{|M|}$ for the minimal inconsistent knowledge base M captures the intuition that the more formulas in a knowledge base are needed to cause inconsistency in it, the less inconsistent it is [24].

Definition 4.2 (*Weighted MinInc inconsistency value* [20]). Let K be a knowledge base. Then the weighted MinInc inconsistency value for K, denoted as MIV_W, is defined as follows:

$$\forall \alpha \in K, \quad \mathsf{MIV}_\mathsf{W}(K, \alpha) = \sum_{M \in \mathsf{MI}(K) s.t. \alpha \in M} \frac{\mathsf{I}_\mathsf{W}(M)}{|M|} = \sum_{M \in \mathsf{MI}(K) s.t. \alpha \in M} \frac{1}{|M|^2}.$$

The weighted MinInc inconsistency value captures the blame on a formula for inconsistency in a knowledge base. The basic idea of the weighted MinInc inconsistency value is that inconsistency in a minimal inconsistent subset is equally shared among all the formulas in it, i.e., each formula in the minimal inconsistent subset has the same blame for inconsistency in it. Furthermore, the blame on a formula for inconsistency in a knowledge base is the sum of the blames on the formula for inconsistencies in all the minimal inconsistent subsets that the formula belongs to. Actually, as shown in [20], both the weighted MI inconsistency measure and the weighted MinInc inconsistency value possess the intuitive properties of inconsistency measures. For example, the weighted MI inconsistency measure satisfies the following three intuitive properties presented in [22]:

- *Consistency*: $\mathsf{I}_\mathsf{W}(K) = 0$ if and only if K is consistent.
- *Monotony w.r.t. MI*: $\mathsf{I}_W(K_1) \leq \mathsf{I}_W(K_2)$ if $\mathsf{MI}(K_1) \subseteq \mathsf{MI}(K_2)$.
- *Free formula independence*: If α is a free formula of $K \cup \{\alpha\}$, then $\mathsf{I}_W(K \cup \{\alpha\}) = \mathsf{I}_W(K)$.

Roughly speaking, the consistency property requires that a desirable inconsistency measure assigns zero only to a consistent knowledge base [22]. The satisfaction of this property ensures that I_W is indeed an inconsistency measure because it can distinguish inconsistent knowledge bases from consistent ones [25]. As a special variant of the Monotony property presented in [22], the monotony w.r.t. MI property states that the degree of inconsistency in a knowledge base increases as

[1] For the sake of simplicity, we abbreviate $\begin{cases} \sum_{\alpha \in A} f(\alpha), & \text{if } A \neq \emptyset, \\ 0, & \text{else.} \end{cases}$ as $\sum_{\alpha \in A} f(\alpha)$.

the number of its minimal inconsistent subsets increases [20]. The Free Formula Independence property states that adding to or deleting from a knowledge a free formula does not change the inconsistency measure for it, since free formulas have nothing to do with the minimal inconsistent subsets of it [22].

The weighted MinInc inconsistency value satisfies the following three intuitive properties:

- *Innocence*: $\forall \alpha \in K, \forall M \in \mathsf{MI}(K), \mathsf{MIV}_{\mathsf{W}}(M, \alpha) = 0$ if $\alpha \notin M$.
- *Fairness*: $\forall \alpha \in K, \forall M \in \mathsf{MI}(K), \mathsf{MIV}_{\mathsf{W}}(M, \alpha) = \frac{1}{|M|}\mathsf{I}_{\mathsf{W}}(M)$ if $\alpha \in M$.
- *Cumulation*: $\mathsf{MIV}_{\mathsf{W}}(K, \alpha) = \sum_{M \in \mathsf{MI}(K)} \mathsf{MIV}_{\mathsf{W}}(M, \alpha)$.

The innocence property states that any formula that is not in a minimal inconsistent subset should not bear any responsibility for the inconsistency in the minimal inconsistent subset [20]. The property of Fairness requires that the blame on each minimal inconsistent subset is shared equally among all the formulas in the minimal inconsistent subset [20]. The cumulation property states that the blame on a formula for the inconsistency in a knowledge base equals to the sum of the blames on this formula for the inconsistencies in all the minimal inconsistent subsets that this formula belongs to [20]. Note that properties of innocence and cumulation together imply that the blame on any free formula in a knowledge base for inconsistency is zero, i.e., any free formula needs not to bear any blame for the inconsistency in the base.

Based on the two measures, we can further define the degree of inconsistency in a requirements specification with regard to a given scenario.

Definition 4.3 (*Degree of inconsistency*). Let $S = \langle S_I, S_E \rangle$ be a scenario and R a requirements specification for S. Then the degree of inconsistency in R w.r.t. S, denoted $\mathsf{I}(R|S)$, is defined as

$$\mathsf{I}(R|S) = \mathsf{I}_{\mathsf{W}}(R \cup S_I).$$

Definition 4.4 (*Blame on a requirement for inconsistency*). Let $S = \langle S_I, S_E \rangle$ be a scenario and R a requirements specification for S. Then the blame on a requirement α in R for inconsistency w.r.t. S, denoted $\mathsf{BI}(R, \alpha|S)$, is defined as

$$\mathsf{BI}(R, \alpha|S) = \mathsf{MIV}_{\mathsf{W}}(R \cup S_I, \alpha).$$

Note that the degree of inconsistency in R w.r.t. S is referred to as the degree of inconsistency in $R \cup S_I$. Correspondingly, the blame on a requirements in R for inconsistency w.r.t. S is the blame on the requirement for the inconsistency in $R \cup S_I$. The properties of the weighted MinInc inconsistency value mentioned above render that $\mathsf{BI}(R, \alpha|S)$ intuitively captures the blame on each requirement for inconsistency in R w.r.t. S.

We use the following examples to illustrate the two measures defined above.

Example 4.1. Given $S = \langle S_I, S_E \rangle$, where $S_I = \{a, b, c\}$ and $S_E = \{d, e, f\}$. Let $R = \{a \to d, b \wedge c \to \neg d, a \wedge d \to e, c \to f, \neg d \to \neg f\}$ be the requirements for S. Clearly,

$$R \cup S_I \vdash d \wedge \neg d, \quad R \cup S_I \vdash f \wedge \neg f.$$

Then R is inconsistent w.r.t. S. The set of minimal inconsistent subsets of $R \cup S_I$ is given as follows:

$$\mathsf{MI}(R \cup S_I) = \{\{a, a \to d, b, c, b \wedge c \to \neg d\}, \{b, c, b \wedge c \to \neg d, \neg d \to \neg f, c \to f\}\}.$$

So

$$\mathsf{I}(R|S) = \frac{2}{5}.$$

The blame on each requirement in R for inconsistencies in R w.r.t. S is given as follows:

$$\mathsf{BI}(R, a \to d|S) = \mathsf{BI}(R, \neg d \to \neg f|S) = \mathsf{BI}(R, c \to f|S) = \frac{1}{25},$$
$$\mathsf{BI}(R, b \wedge c \to \neg d|S) = \frac{2}{25},$$
$$\mathsf{BI}(R, a \wedge d \to e|S) = 0.$$

Note that $a \wedge d \to e$ is a free formula in $R \cup S_I$ and has nothing to do with the inconsistency in $R \cup S_I$. Evidently, $\mathsf{BI}(R, a \wedge d \to e|S) = 0$ captures this fact. In contrast, $b \wedge c \to \neg d$ is the common formula of the two minimal inconsistent subsets, and then it should bear more blame for the inconsistency in $R \cup S_I$. Actually, $\frac{2}{25} > \frac{1}{25}$ describes exactly this intuition.

Example 4.2. Consider Example 2.1 again. R is inconsistent w.r.t. S_1, and $\mathsf{MI}(R \cup S_1) = \{M_1\}$, where

$$M_1 = \{(\forall x)(\neg \mathtt{Aut}(x) \to \neg \mathtt{Ent}(x)), (\forall x)(\mathtt{Eme}(x) \to \mathtt{Ent}(x) \wedge \neg \mathtt{Aut}(x)), \mathtt{Eme}(\mathtt{fi\_en})\}.$$

So

$$\mathsf{I}(R|S_1) = \frac{1}{3}.$$

The blame on each requirement in R for inconsistencies in R w.r.t. S_1 is given as follows:

$$\begin{aligned}
&\mathsf{BI}(R, (\forall x)(\neg \mathtt{Aut}(x) \to \neg \mathtt{Ent}(x))|S_1) = \frac{1}{9},\\
&\mathsf{BI}(R, (\forall x)(\mathtt{Eme}(x) \to \mathtt{Ent}(x) \wedge \neg \mathtt{Aut}(x))|S_1) = \frac{1}{9},\\
&\mathsf{BI}(R, (\forall x)(\mathtt{Aut}(x) \to \mathtt{Ent}(x))|S_1) = \mathsf{BI}(R, (\forall x)(\neg \mathtt{Aut}(x) \to \mathtt{Ala}(x))|S_1) = 0,\\
&\mathsf{BI}(R, (\forall x)(\mathtt{Ala}(x) \to \neg \mathtt{Act}(x, \mathtt{entr}))|S_1) = 0,\\
&\mathsf{BI}(R, (\forall x)(\neg \mathtt{Aut}(x) \wedge \neg \mathtt{Eme}(x) \to \neg \mathtt{Ent}(x))|S_1) = 0,\\
&\mathsf{BI}(R, (\forall x)(\neg \mathtt{Aut}(x) \wedge \neg \mathtt{Eme}(x) \to \neg \mathtt{Act}(x, \mathtt{entr}))|S_1) = 0.
\end{aligned}$$

4.2. Measuring incomplete requirements

To revise incomplete requirements, developers often need to know how incomplete a requirements specification is with regard to a given scenario. Let us recall the definition of incompleteness. A requirements specification R is incomplete with regard to a given scenario S if some expected behaviors in the scenario cannot be derived from the requirements specification. Intuitively, measures for the degree of incompleteness in a requirements specification w.r.t. a given scenario should at least distinguish it from the case in which all the expected behaviors in the scenario can be derived from the requirements specification, i.e, $R \cup S_I \vdash S_E$. On the other hand,note that if $R \cup S_I \vdash \bot$ for S, then $R \cup S_I \vdash S_E$ because of trivial reasoning in the presence of inconsistency. However, in this case, $R \cup S_I \vdash S_E$ does not necessarily mean that R is complete with regard to S. We therefore need to distinguish inconsistent requirements from truly complete requirements with regard to a given scenario. To address this issue,we first define the drastic measure for the degree of incompleteness in a requirements specification with regard to a given scenario.

Definition 4.5 (*Drastic measure*)**.** Let $S = \langle S_I, S_E \rangle$ be a scenario and R a requirements specification for S. Then the drastic measure for the degree of incompleteness in R w.r.t. S, denoted as $\mathsf{C_d}(R|S)$, is defined as

$$\mathsf{C_d}(R|S) = \begin{cases} -1, & R \cup S_I \vdash \bot, \\ 0, & R \cup S_I \nvdash \bot, R \cup S_I \vdash S_E, \\ 1, & \exists \alpha \in S_E\ s.t.\ R \cup S_I \nvdash \alpha. \end{cases}$$

Note that the drastic measure $\mathsf{C_d}$ allows us only to distinguish among inconsistent, complete and incomplete requirements specifications with regard to a given scenario. It cannot be used to distinguish between different incomplete requirements specifications with regard to a given scenario.

However, for it to be more intuitive and more distinctive, the measure for the degree of incompleteness should take into account the following aspects:

- It can distinguish consistent requirements specifications from inconsistent ones w.r.t. a given scenario. As mentioned above, it is meaningless to measure how incomplete an inconsistent requirements specification is with the given scenario because of trivial reasoning in the presence of inconsistency. We need only a designated value to differentiate inconsistent requirements specifications from consistent ones for a given scenario.
- For all consistent requirements specifications w.r.t. a given scenario, an intuitive measure for the degree of incompleteness should distinguish truly complete requirements specifications from incomplete ones for the scenario.
- An intuitive measure for the degree of incompleteness should assign the same value to all truly complete requirements specifications w.r.t. a given scenario. That is, it cannot be used to differentiate truly complete requirements specifications from each other. It is necessary to ensure that an individual measure is indeed a measure for incompleteness.

- The measure for the degree of incompleteness should take into account the expected behaviors of the system-to-be that could not be derived from a requirements specification in a given scenario. Intuitively, given a scenario S, the incompleteness of a requirement specification R w.r.t. the scenario depends on the number of missing requirements w.r.t. S, which is captured implicitly by the expected behaviors of the system-to-be that could not be derived from the requirements specification in the scenario. Moreover, the more expected behaviors that could not be derived from a requirements specification in the scenario, the more incomplete the requirements specification w.r.t. the scenario is.

To address these issues, we take into account the expected responses that could not be derived from a requirements specification in a scenario to measure the degree of incompleteness of the requirements specification. Moreover, we call such expected responses the set of missing expected responses.

Definition 4.6 (*Missing expected responses*)**.** Let $S = \langle S_I, S_E\rangle$ be a scenario and R a consistent requirements specification for S. Then the set of missing expected responses induced by R in S, denoted as $\mathsf{M}E(R|S)$, is defined as the expected responses that could not be derived from R in S, i.e.,

$$\mathsf{M}E(R|S) = \{\alpha \in S_E |\ R \cup S_I \nvdash \alpha\}.$$

We use the normalized number of the missing expected responses induced by a requirements specification in a scenario to measure the degree in incompleteness of the requirements specification w.r.t. the scenario.

Definition 4.7 (*Degree of incompleteness*)**.** Let $S = \langle S_I, S_E\rangle$ be a scenario and R a requirements specification for S. Then the degree of incompleteness in R w.r.t. S, denoted as $\mathsf{C}(R|S)$, is defined as

$$\mathsf{C}(R|S) = \begin{cases} -1, & R \cup S_I \vdash \perp, \\ \frac{|\mathsf{M}E(R|S)|}{|S_E|}, & R \cup S_I \nvdash \perp. \end{cases}$$

As the drastic measure, -1 is considered as the designated value for inconsistent requirements specifications, i.e., it satisfies the following property of inconsistency:

- *Inconsistency*: $\mathsf{C}(R|S) = -1$ if and only if R is *inconsistent* w.r.t. S.

Moreover, the measure for the degree of incompleteness satisfies the following property of completeness:

- *Completeness*: $\mathsf{C}(R|S) = 0$ if and only if R is consistent and complete w.r.t. S.

This property ensures that C is indeed an incompleteness measure, because it could not distinguish between different consistent and complete requirements specifications w.r.t. a given scenario but could measure incomplete ones.

Compared to the drastic measure, the measure $\mathsf{C}(R|S)$ provides a finer-grained description of incompleteness in a requirements specification R w.r.t. the given scenario S. Given a scenario, the higher the number of missing expected responses induced by a requirements specification, the more incomplete the requirements specification is, i.e.,

- *Monotony w.r.t. ME*: for any two consistent requirements specifications R and R' for S, $\mathsf{C}(R|S) \leq \mathsf{C}(R'|S)$ if $|\mathsf{M}E(R|S)| \leq |\mathsf{M}E(R'|S)|$.

Example 4.3. Given $S = \langle S_I, S_E\rangle$, where $S_I = \{a, b\}$ and $S_E = \{c, d, e, f\}$. Let $R_0 = \{a \to c, c \wedge b \to d\}$, then

$$R_0 \cup S_I \nvdash e, \text{ and } R_0 \cup S_I \nvdash f.$$

So

$$\mathsf{C}_{\mathrm{d}}(R_0|S) = 1.$$

Suppose that after revision, a new requirements specification R_1 is given as follows: $R_1 = \{a \to c, c \wedge b \to d, c \wedge d \to e\}$. Then

$$R_1 \cup S_I \nvdash f.$$

So

$$\mathsf{C}_{\mathrm{d}}(R_1|S) = 1.$$

Note that the two drastic measures $\mathsf{C}_d(R_0|S)$ and $\mathsf{C}_d(R_1|S)$ only say that the two corresponding requirements specifications are incomplete with regard to S. Intuitively, R_1 is less incomplete than R_0 with regard to S. However, the weighted measure can make this distinction, since

$$\mathsf{C}(R_1|S) = \frac{1}{4} < \mathsf{C}(R_0|S) = \frac{1}{2}.$$

Example 4.4. Consider Example 2.1 again. R is incomplete w.r.t. $S_3 = \langle S_{3_I}, S_{3_E} \rangle$ where

$$S_{3_I} = \{\neg \mathtt{Aut}(\mathtt{vis3}), \neg \mathtt{Eme}(\mathtt{vis3})\}, \quad S_{3_E} = \{\neg \mathtt{Ent}(\mathtt{vis3}), \mathtt{Act}(\mathtt{vis3}, \mathtt{help})\}.$$

And

$$\mathsf{C}(R|S_3) = \frac{|\{\mathtt{Act}(\mathtt{vis1}, \mathtt{help})\}|}{|S_{3_E}|} = \frac{1}{2}.$$

4.3. Measuring redundant requirements

An appropriate proposal for measuring redundant requirements should focus on measuring the degree of redundancy in a requirements specification w.r.t. a scenario as well as the blame on each requirement for the redundancy. Roughly speaking, a measure for the degree of redundancy in a requirements specification describes how redundant the whole requirements specification is w.r.t. a scenario, whilst the blame on each requirement for the redundancy captures the contribution made by each requirement to the redundancy in the specification.

Intuitively, to assess the amount of redundant information in a requirements specification w.r.t. a scenario, we need to first identify simplifications of the specification that exactly convey necessary and sufficient information about the scenario. We start the measure for the degree of redundancy in a requirements specification from the minimal simplifications of a requirements specification.

Definition 4.8 (*Minimal simplifications* [13])**.** Let $S = \langle S_I, S_E \rangle$ be a scenario and R a requirements specification for S. A subset R' of R is called a minimal simplification of R w.r.t. S if

(1) R' is a simplification of R w.r.t. S.
(2) no proper subset of R' is a simplification of R w.r.t. S.

In particular, we use $\mathsf{MS}(R|S)$ to denote the set of minimal simplifications of R w.r.t. S.

Essentially, each minimal simplification of a redundant requirements specification conveys adequate requirements information w.r.t. a given scenario. Any requirement not belonging to a minimal simplification can be considered as potentially redundant because we can derive all the expected responses from the minimal simplification in the scenario. That is, each minimal simplification of a requirements specification provides a perspective to separate the specification into two parts, i.e., a minimal set of requirements conveying adequate information and potential redundancy. Moreover, the smaller the size of a minimal simplification, the bigger the number of potential redundant requirements is. To address this issue, we build a measure for the degree of redundancy for a redundant requirements specification upon the ratio of the number of requirements to the smallest number of requirements that convey adequate information. Evidently, such measures capture how far a redundant requirements specification is beyond the most adequate requirements specification. On the other hand, for a non-redundant requirements specification, its incompleteness essentially describes how far the requirements specification is from an expected adequate (i.e., truly complete but not redundant) requirements specification. So, we may build redundancy measures for a non-redundant requirements specification upon its incompleteness assessment.

Definition 4.9 (*Degree of redundancy*)**.** Let $S = \langle S_I, S_E \rangle$ be a scenario and R a requirements specification for S. Then the degree of redundancy in R w.r.t. S, denoted as $\mathsf{D}_r(R|S)$, is defined as

$$\mathsf{D}_r(R|S) = \begin{cases} \max\left\{\frac{|R|}{|R'|} \,\middle|\, R' \in \mathsf{MS}(R|S)\right\}, & \mathsf{MS}(R|S) \neq \emptyset, \\ 1 - \mathsf{C}(R|S), & \mathsf{MS}(R|S) = \emptyset, \text{ and } \mathsf{C}(R|S) \geq 0, \\ -1, & \mathsf{MS}(R|S) = \emptyset, \text{ and } \mathsf{C}(R|S) = -1. \end{cases}$$

Roughly speaking, -1 is still considered as a designated value for inconsistent but non-redundant requirements specifications w.r.t. a given scenario. Moreover, for redundant R w.r.t. S, $\mathsf{D}_r(R|S)$ aims to capture the most redundant perspective of R w.r.t. S, that is, $\mathsf{D}_r(R|S)$ focuses on the case of the smallest minimal simplification. In addition, we consider $1 - \mathsf{C}(R|S)$ as the degree of redundancy for consistent but incomplete requirements specification R w.r.t. S.

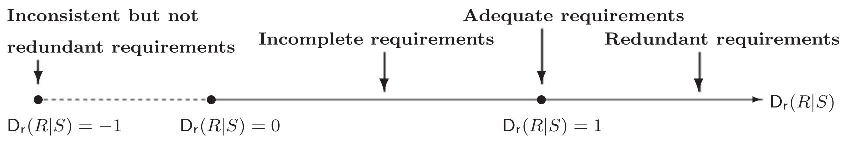

Fig. 2. From Inconsistency to Redundancy

As shown by the following proposition, $\mathrm{D_r}(R|S)$ may be considered as an intuitive measure for the degree of redundancy.

Proposition 4.1. *Let $S = \langle S_I, S_E\rangle$ be a scenario and R a requirements specification for S. Then $\mathrm{D_r}$ satisfies the following properties:*

- *Monotony: If R is redundant w.r.t. S, then $\mathrm{D_r}(R|S) \le \mathrm{D_r}(R \cup R'|S)$ for any R'.*
- *Redundancy: $\mathrm{D_r}(R|S) > 1$ if R is redundant w.r.t. S.*
- *Adequacy: $\mathrm{D_r}(R|S) = 1$ if and only if $R \cup S_I \nvdash \bot$, $R \cup S_I \vdash S_E$, and $\forall R' \subset R$, $R' \cup S_I \nvdash S_E$.*
- *Incompleteness: $0 \le \mathrm{D_r}(R|S) < 1$ if R is consistent but incomplete with regard to S.*

Proof. We only provide a proof for Monotony. Proofs for the other three properties are evident.

- *Monotony*. Let R be a redundant requirements specification w.r.t. S. Then $\mathrm{M}S(R|S) \neq \emptyset$ and $\mathrm{M}S(R|S) \subseteq \mathrm{M}S(R \cup R'|S)$ for any R'. So,

$$\begin{aligned}\mathrm{D_r}(R|S) &= \max\left\{\frac{|R|}{|R''|}|R'' \in \mathrm{M}S(R|S)\right\}\\ &\le \max\left\{\frac{|R \cup R'|}{|R''|}|R'' \in \mathrm{M}S(R|S)\right\}\\ &\le \max\left\{\frac{|R \cup R'|}{|R''|}|R'' \in \mathrm{M}S(R \cup R'|S)\right\}\\ &= \mathrm{D_r}(R \cup R'|S). \quad \Box\end{aligned}$$

Intuitively, the monotony property states that as a redundant requirements set w.r.t. a given scenario expands, the degree of redundancy in it cannot decrease. The redundancy property states that the degree of redundancy in R w.r.t. S should be strictly greater than 1 if R is redundant w.r.t. S. Essentially, it implies that there is at least one perspective to separate R into two nonempty parts, i.e., a minimal simplification and a set of potentially redundant requirements. In contrast, the adequacy property states that R with the degree of redundancy 1 w.r.t. S is exactly an adequate requirements specification for S, i.e., it is consistent and complete, but not redundant w.r.t. S. The incompleteness property states that the degree of redundancy in R w.r.t. S is a non-negative value less than 1 if R is consistent but incomplete w.r.t. S.

Informally speaking, incompleteness is due to insufficient information about requirements. By contrast, both redundancy and inconsistency result from too much information. However, redundancy requires that at least one proper subset of a requirements specification conveys adequate information about the given scenario, i.e., a redundant requirements specifications w.r.t. a scenario conveys adequate information about the given scenario as well as other extra information. But this does not necessarily hold for inconsistency, because inconsistency focuses only on contradiction in a requirements specification rather than the adequacy of the specification w.r.t. a scenario. Furthermore, if a requirements specification is inconsistent but not redundant, it is also meaningless to assess the incompleteness because of trivial reasoning in the presence of inconsistency. Actually, the last three properties support that $\mathrm{D_r}$ distinguishes between redundant requirements ($\mathrm{D_r}(R|S) > 1$), adequate requirements (i.e., consistent, complete but non-redundant requirements, $\mathrm{D_r}(R|S) = 1$), incomplete requirements ($0 \le \mathrm{D_r}(R|S) < 1$), and inconsistent but non-redundant requirements ($\mathrm{D_r}(R|S) = -1$) for a given scenario within the context of redundancy. This is illustrated in Fig. 2. In some sense, this distinction shows how the dimensions of inconsistency, incompleteness and redundancy are brought together in a linear scale. However, inconsistency, incompleteness, and redundancy are essentially heterogeneous. We have to point out that there are fewer possibilities for algebraic manipulations concerning this linear scale, except making the distinction as described above.

Example 4.5. Given $S = \langle S_I, S_E\rangle$, where $S_I = \{a, b\}$ and $S_E = \{c, d\}$. Let $R = \{a \to c, c \wedge b \to d, a \wedge b \to (\neg c \vee d)\}$, then

$$\mathrm{M}S(R|S) = \{R_1, R_2\},$$

where $R_1 = \{a \to c, c \wedge b \to d\}$, $R_2 = \{a \to c, a \wedge b \to (\neg c \vee d)\}$. So,

$$\mathrm{D_r}(R|S) = \max\left\{\frac{|R|}{|R_1|}, \frac{|R|}{|R_2|}\right\} = \frac{3}{2}.$$

Example 4.6. Consider Example 4.3 again.

$$0 < \mathrm{D_r}(R_0|S) = \frac{1}{2} < \mathrm{D_r}(R_1|S) = \frac{3}{4} < 1.$$

This result is intuitive, since we know that both R_1 and R_0 are incomplete w.r.t. S, but R_1 is less incomplete than R_0 w.r.t. S.

Definition 4.10 (*Maximal potential redundancy*)**.** Let $S = \langle S_I, S_E\rangle$ be a scenario and R a requirements specification for S. A subset R' of R is called a maximal potential redundancy of R w.r.t. S if $R \setminus R'$ is a minimal simplification of R w.r.t. S.

We use $\mathrm{MR}(R|S)$ to denote the set of maximal potential redundancies of R w.r.t. S. In particular, we use $\mathrm{MR_{MAX}}(R|S)$ to denote the set of the largest maximal potential redundancies of R w.r.t. S, that is,

$$\mathrm{MR_{MAX}}(R|S) = \{R'|R' \in \mathrm{MR}(R|S),\ \text{and}\ \forall R'' \in \mathrm{MR}(R|S), |R''| \leq |R'|\}.$$

Note that if R' is one of the largest maximal potential redundancies of R w.r.t. S, then $R \setminus R'$ must be the smallest minimal simplification of R w.r.t. S.

Example 4.7. Consider Example 4.5 again. Both $R'_1 = \{a \wedge b \to (\neg c \vee d)\}$ and $R'_2 = \{c \wedge b \to d\}$ are the largest maximal potential redundancies of R w.r.t. S.

As mentioned above, each minimal simplification of a requirements specification provides a possible perspective to separate the requirements specification into two parts, i.e., the minimal simplification and the corresponding maximal potential redundancy. Moreover, each maximal potential redundancy may be considered as one of the perspectives to capture redundant information in the requirements specification w.r.t. a scenario. Intuitively, from the perspective of a given maximal potential redundancy, the degree of redundancy in a requirements specification should be equally shared by each requirement belonging to the maximal potential redundancy. Similar to the degree of redundancy, we focus on the blame on a requirement for the largest maximal potential redundancies (i.e., the most redundant perspectives of a requirements specification) that the requirement belongs to.

Definition 4.11 (*Blame on a requirement for redundancy*)**.** Let $S = \langle S_I, S_E\rangle$ be a scenario and R a requirements specification for S. The blame on a requirement α in R for redundancy in R w.r.t. S, denoted $\mathrm{BR}(R, \alpha|S)$, is defined as

$$\mathrm{BR}(R, \alpha|S) = \begin{cases} \frac{1}{|R'|} \times \frac{|R|}{|R\setminus R'|}, & \exists R' \text{ s.t. } \alpha \in R' \text{ and } \frac{|R|}{|R\setminus R'|} = \max\limits_{R'' \in \mathrm{MR}(R|S) s.t. \alpha \in R''} \left\{\frac{|R|}{|R\setminus R''|}\right\}, \\ 0, & \text{else.} \end{cases}$$

Intuitively, $\mathrm{BR}(R, \alpha|S)$ captures the contribution of α to the most redundant perspective in R w.r.t. S in which α is involved.

Evidently, we can get the following proposition from Definition 4.11.

Proposition 4.2. *Let $S = \langle S_I, S_E\rangle$ be a scenario and R a requirements specification for S. $\mathrm{BR}(R, \alpha|S)$ satisfies the following properties:*

- *Independence:* $\mathrm{BR}(R, \alpha|S) = 0$ *if and only if* $\alpha \notin R'$ *for each* $R' \in \mathrm{MR}(R|S)$.
- *Fairness: If* $\exists \alpha, \beta \in R$ *s.t.* $\forall R \in \mathrm{MR}(R|S)$, $\alpha \in R'$ *iff* $\beta \in R'$, *then*

$$\mathrm{BR}(R, \alpha|S) = \mathrm{BR}(R, \beta|S).$$

- *Distribution: For* $\mathrm{D_r}(R|S) > 1$, $\sum\limits_{\alpha \in R'} \mathrm{BR}(R, \alpha|S) = \mathrm{D_r}(R|S)$ *if* $R' \in \mathrm{MR}_{MAX}(R|S)$.

Proof. The properties of Independence and Fairness are evident. We only provide a proof for Distribution.

- *Distribution.* Let R be a requirements specification for S such that $\mathrm{D_r}(R|S) > 1$. Then $\mathrm{M}S(R|S) \neq \emptyset$ and

$$\begin{aligned} \mathrm{D_r}(R|S) &= \max\left\{\frac{|R|}{|R''|} | R'' \in \mathrm{M}S(R|S)\right\} \\ &= \max\left\{\frac{|R|}{|R \setminus R'''|} | R''' \in \mathrm{M}R(R|S)\right\}. \end{aligned}$$

Let $R' \in \mathsf{MR}_{\mathsf{MAX}}(R|S)$, then

$$\frac{|R|}{|R \setminus R'|} = \max\left\{\frac{|R|}{|R \setminus R'''|}|R''' \in \mathsf{M}R(R|S)\right\}.$$

So, $\mathsf{D_r}(R|S) = \frac{|R|}{|R \setminus R'|}$.
On the other hand, $\forall \alpha \in R'$,

$$\begin{aligned}\mathsf{B}R(R,\alpha|S) &= \max_{R'' \in \mathsf{MR}(R|S) s.t. \alpha \in R''}\left\{\frac{|R|}{|R \setminus R''|}\right\} \times \frac{1}{|R''|}\\ &= \frac{|R|}{|R \setminus R'|} \times \frac{1}{|R'|}.\end{aligned}$$

Obviously,

$$\sum_{\alpha \in R'} \mathsf{BR}(R,\alpha|S) = \frac{|R|}{|R \setminus R'|} = \mathsf{D_r}(R|S). \quad \square$$

Intuitively, the property of independence states that a requirement that does not belong to any maximal potential redundancy of R takes no responsibility for redundancy in R, i.e., such a requirement that is free from redundancy should be assigned zero. The property of fairness states that two formulas that belong to the same maximal potential redundancy should bear the same blame for the redundancy of the maximal potential redundancy. The property of distribution states that the blame for the redundancy in a redundant R w.r.t. S may be shared among all the requirements that belong to one of the largest maximal potential redundancies in R. It coincides with the intention of $\mathsf{D_r}(R|S)$ to capture the most redundant perspective of R w.r.t. S. In this sense, Proposition 4.2 shows that $\mathsf{BR}(R,\alpha|S)$ intuitively captures the blame on each requirement for the redundancy.

Example 4.8. Consider Example 4.5 again. Then

$$\begin{aligned}&\mathsf{BR}(R, a \wedge b \to (\neg c \vee d)|S) = \mathsf{BR}(R, c \wedge b \to d|S) = \frac{3}{2},\\ &\mathsf{BR}(R, a \to c|S) = 0.\end{aligned}$$

This result is intuitive, since $a \to c$ does not belong to any maximal potential redundancy of R w.r.t. S.

Example 4.9. Consider Example 2.1 again. R is redundant w.r.t. S_2, and

$$\begin{aligned}\mathsf{MS}(R|S_2) = \{\{&(\forall x)(\mathtt{Aut}(x) \to \mathtt{Ent}(x)), (\forall x)(\mathtt{Eme}(x) \to \mathtt{Ent}(x) \wedge \neg\mathtt{Aut}(x)),\\ &(\forall x)(\neg\mathtt{Aut}(x) \wedge \neg\mathtt{Eme}(x) \to \neg\mathtt{Ent}(x))\}\}.\end{aligned}$$

So

$$\mathsf{D_r}(R|S_2) = \frac{7}{3}.$$

The blame on each requirement in R for the redundancy in R w.r.t. S_2 is given as follows:

$$\mathsf{BR}(R, (\forall \mathtt{x})(\mathtt{Aut(x)} \to \mathtt{Ent(x)})|S_2) = 0,$$
$$\mathsf{BR}(R, (\forall \mathtt{x})(\mathtt{Eme(x)} \to \mathtt{Ent(x)} \wedge \neg\mathtt{Aut(x)})|S_2) = 0,$$
$$\mathsf{BR}(R, (\forall \mathtt{x})(\neg\mathtt{Aut(x)} \wedge \neg\mathtt{Eme(x)} \to \neg\mathtt{Ent(x)})|S_2) = 0,$$
$$\mathsf{BR}(R, (\forall \mathtt{x})(\neg\mathtt{Aut(x)} \to \neg\mathtt{Ent(x)})|S_2) = \frac{7}{12},$$
$$\mathsf{BR}(R, (\forall \mathtt{x})(\neg\mathtt{Aut(x)} \to \mathtt{Ala(x)})|S_2) = \frac{7}{12},$$
$$\mathsf{BR}(R, (\forall \mathtt{x})(\neg\mathtt{Aut(x)} \wedge \neg\mathtt{Eme(x)} \to \neg\mathtt{Ent(x)})|S_2) = \frac{7}{12},$$
$$\mathsf{BR}(R, (\forall \mathtt{x})(\neg\mathtt{Aut(x)} \wedge \neg\mathtt{Eme(x)} \to \neg\mathtt{Act(x, entr)})|\mathtt{S}_2) = \frac{7}{12}.$$

Note that all the measures presented above are syntax sensitive. That is, it is not necessary that any two logically equivalent knowledge bases have the same measurement. For example, $\mathsf{I}_W(\{a, \neg a\}) = \frac{1}{2}$ but $\mathsf{I}_W(\{a \wedge \neg a\}) = 1$. However, this syntax sensitivity is meaningful for measuring non-canonical requirements. As mentioned earlier, formulating requirements is syntax sensitive, and any two different knowledge bases (e.g., $\{a, \neg a\}$ and $\{a \wedge \neg a\}$) are considered as different sets of requirements [17]. In this sense, $\{a \wedge \neg a\}$ contains only one self-contradictory requirement statement, whilst $\{a, \neg a\}$ contains two self-consistent requirements, i.e., a and $\neg a$, but two requirements contradict each other. So it is not surprising that $\{a \wedge \neg a\}$ and $\{a, \neg a\}$ have different measures.

5. Generating proposals for handling non-canonical requirements

As mentioned earlier, a general framework that is feasible for handling non-canonical requirements should provide an approach to identifying which requirements should be changed rather than how to change them. Roughly speaking, a feasible approach to generating proposals for handling non-canonical requirements can be divided into three steps:

(1) to generate preliminary proposals according to the blame on each requirement for the inconsistency;
(2) to evaluate the preliminary proposals on their impact on incompleteness in the requirements specification, and then to revise the preliminary proposals;
(3) to evaluate the revised proposals on their impact on the redundancy in the requirements specification, and then to adapt these proposals to simplify the requirements specifications.

In this paper, a proposal for handling non-canonical requirements consists of three parts, i.e., the set of requirements to be revised, the set of requirements to be suspended,[2] and the set of expected responses for inducing new requirements. Roughly speaking, the set of requirements to be revised corresponds to strategies for resolving inconsistency in the requirements specification w.r.t. a given scenario, whilst the set of requirements to be suspended corresponds to strategies for simplifying the requirements specification w.r.t. a given scenario. In contrast, the set of expected responses for inducing new requirements is exactly the set of expected responses that cannot be derived from the (original or revised) requirements specification in the given scenario. It is considered as a hint on missing requirements about the scenario, and then corresponds to strategies for resolving incompleteness of the requirements specification w.r.t. the given scenario.

Roughly speaking, given a scenario, each requirement in the minimal inconsistent subsets may be considered as potential one to be revised, and each requirement in the maximal potential redundancies of a requirements specification may be considered as one to be suspended. On the other hand, each expected response that cannot be derived from the unchanged requirements may be considered as one to be included. Along this line, we define the following (valid) potential proposals for handling non-canonical requirements.

Definition 5.1 (*Potential proposals*). Let $S = \langle S_I, S_E \rangle$ be a scenario and R a requirements specification for S. A potential proposal for handling non-canonical requirements in R w.r.t. S, denoted as P, is defined as a 3-tuple $P = \langle P_\odot, P_\oplus, P_\ominus \rangle$ such that

(1) $P_\odot \subseteq \bigcup_{R' \in \mathsf{M}I(R \cup S_I)} R'$,
(2) $P_\ominus \subseteq \bigcup_{R' \in \mathsf{M}R(R|S)} R'$,
(3) $P_\oplus \subseteq \mathsf{M}E(R \setminus (P_\odot \cup P_\ominus)|S)$ if $R \setminus (P_\odot \cup P_\ominus)$ is consistent w.r.t. S.

Note that in a proposal for handling non-canonical requirements $P = \langle P_\odot, P_\oplus, P_\ominus \rangle$, $P_\odot$, $P_\oplus$, and $P_\ominus$ are the set of requirements to be revised, the set of expected responses to be included, and the set of requirements to be suspended,

[2] Allowing for frequent changes in requirements engineering, developers often suspend rather than remove such requirements.

respectively. In particular, for an adequate requirements specification R for S, $\mathsf{M}I(R|S) = \mathsf{M}R(R|S) = \mathsf{M}E(R|S) = \emptyset$, so $P_{\odot} = P_{\oplus} = P_{\ominus} = \emptyset$, i.e., there is no need to revise R. This result is intuitive.

Definition 5.2 (*Valid potential proposals*)**.** Let $S = \langle S_I, S_E \rangle$ be a scenario and R an inconsistent requirements specification for S. Let P be a potential proposal for handling non-canonical requirements. Then P is valid w.r.t. S if

(1) $(R \cup P_{\oplus} \setminus (P_{\odot} \cup P_{\ominus})) \cup S_I \nvdash \bot$.
(2) $(R \cup P_{\oplus} \setminus (P_{\odot} \cup P_{\ominus})) \cup S_I \vdash S_E$.

Essentially, condition (1) ensures that the revised requirements specification should be consistent w.r.t. the given scenario, and condition (2) requires that the revised requirements specification should be also complete w.r.t. the given scenario.

Example 5.1. Given $S = \langle S_I, S_E \rangle$, where $S_I = \{a, b, c\}$ and $S_E = \{d, e, f, g\}$. Let $R = \{a \to d, b \wedge c \to e, a \wedge c \to f, e \to f, b \to \neg d \wedge g, \neg d \to \neg e\}$ be the requirements about S. Then

$$R \cup S_I \vdash d \wedge \neg d,\ R \cup S_I \vdash e \wedge \neg e.$$

And

$$\mathsf{MI}(R \cup S_I) = \{\{a, b, a \to d, b \to \neg d \wedge g\}, \{b, c, b \to \neg d \wedge g, b \wedge c \to e, \neg d \to \neg e\}\}.$$

Obviously, $P = \langle P_{\ominus}, P_{\odot}, P_{\oplus} \rangle$ is a valid potential proposal w.r.t. S, where

$$P_{\ominus} = \{e \to f\}, \quad P_{\odot} = \{b \to \neg d \wedge g\}, \quad P_{\oplus} = \{g\}.$$

Sometimes developers also need to prioritize valid potential proposals in different ways to identify the most appropriate proposals for handling non-canonical software requirements. Note that inconsistency handling is central to non-canonical requirements management. So, we need to associate priority with the appropriateness of a valid potential proposal for handling inconsistency. Intuitively, to evaluate the appropriateness of a valid potential proposal, the following aspects should be taken into account:

- First, the requirements with higher blame on inconsistency should be given the priority to be included in the set of requirements to be revised. Suppose that $P_{1\odot}$ and $P_{2\odot}$ are two sets of requirements to be revised for handling inconsistency in R w.r.t. S, then $P_{1\odot}$ *is more appropriate than* $P_{2\odot}$, denoted by $P_{1\odot} \preceq P_{2\odot}$, if $\forall M \in \mathsf{MI}(R \cup S_I)$, $\exists \alpha \in P_{1\odot} \cap M$ such that $BI(R, \alpha|S) \geq BI(R, \beta|S)$ for all $\beta \in P_{2\odot} \cap M$. That is, for each minimal inconsistent subset, the more appropriate proposal should include at least one requirement in the minimal inconsistent subset such that its blame on inconsistency is higher than that of any requirements in the subset involved in the less appropriate proposal.
- Second, the smaller set of expected responses to be included is preferred. Inconsistency handling actions may lead to incompleteness or redundancy in many cases. Note that the smaller the set of expected responses to be included, the less incomplete the revised requirements specification due to inconsistency handling is.
- Third, the minimal simplification with smaller size is preferred. Informally speaking, each minimal simplification conveys the same information as the original redundant requirements specification, and provides a possible way to simplify the redundant requirements specification. Moreover, the smaller the size of a minimal simplification, the more concise the minimal simplification is.

Definition 5.3 (*More appropriateness*)**.** Let $S = \langle S_I, S_E \rangle$ be a scenario and R an inconsistent requirements specification for S. Let P_1 and P_2 be two valid potential proposals. P_1 is more appropriate than P_2 for handling non-canonical requirements in R w.r.t. S if

(1) $P_{1\odot} \preceq P_{2\odot}$, or
(2) $|P_{1\oplus}| \leq |P_{2\oplus}|$ if $P_{1\odot} \simeq P_{2\odot}$,[3] or
(3) $|P_{1\ominus}| \geq |P_{2\ominus}|$ if $|P_{1\oplus}| = |P_{2\oplus}|$ and $P_{1\odot} \simeq P_{2\odot}$.

Note that condition (1) ensures that the requirements with higher blame for inconsistency should be given the priority to be included in the more appropriate proposals, whilst conditions (2) and (3) state that the the smaller set of expected responses to be included and the smaller minimal simplification are preferred, respectively.

We present an algorithm for generating proposals which support this intuition.

Our algorithm for generating proposals consists of three sub-algorithms, i.e., the algorithm for generating $P_{\odot}$, the algorithm for generating $P_{\oplus}$ given $P_{\odot}$, and the algorithm for generating $P_{\ominus}$ given $P_{\odot}$ and $P_{\oplus}$. We start with the algorithm for generating proposals for handling inconsistency.

[3] $P_{1\odot} \simeq P_{2\odot}$ if $P_{1\odot} \preceq P_{2\odot}$ and $P_{2\odot} \preceq P_{1\odot}$.

For each $R' \subseteq R$, let $\mathsf{BI}_{\mathsf{MAX}}(R')$ be the set of requirements of R' with the highest degree of blame for inconsistency in R, i.e.,

$$\mathsf{BI}_{\mathsf{MAX}}(R') = \{\alpha|\alpha \in R', \text{ and } \forall\beta \in R', \mathsf{BI}(R, \beta|S) \leq \mathsf{BI}(R, \alpha|S)\}.$$

Let Q be a set of subsets of R, then we abbreviate $\bigcup_{K' \in Q} K'$ as $\bigcup Q$.

An algorithm for generating proposals $P_\odot$ for handling inconsistency based on the degree of blame is given in Algorithm 1.

Algorithm 1. GENERATE_PROPOSALS_FOR_INCONRE(S, R)
Input: A Scenario S and the related requirements specification R
Output: A set of proposals $\Pi_\odot$

```
1  $\Pi_\odot \leftarrow \emptyset$
2  $\Pi_{\odot 0} \leftarrow \{\emptyset\}$
3  $T \leftarrow 0$
4  $Q \leftarrow \mathsf{MI}(R \cup S_I)$
5  while $Q \neq \emptyset$
6     do $\Pi_{\odot T+1} \leftarrow \emptyset$
7        for each $P_\odot \in \Pi_{\odot T}$
8           $Q \leftarrow \mathsf{MI}(R \cup S_I) \setminus \{M|M \in \mathsf{MI}(R \cup S_I), M \cap P_\odot \neq \emptyset\}$
9           for each $\alpha \in \mathsf{BI}_{\mathsf{MAX}}(R \cap \bigcup Q)$
10             $P_\odot \leftarrow P_\odot \cup \{\alpha\}$
11             $\Pi_{\odot T+1} \leftarrow \Pi_{\odot T+1} \cup \{P_\odot\}$
12       $T \leftarrow T + 1$
13 $\Pi_\odot \leftarrow \Pi_{\odot T}$
14 return $\Pi_\odot$
```

Roughly speaking, this algorithm focuses on finding proposals that consist of some requirements with the highest blame on inconsistency for each minimal inconsistent subset of a requirements specification in a scenario. Lines 8–10 ensure that the requirement chosen for each $P_\odot$ in each round is one of the requirements with the highest blame on inconsistency in all the minimal inconsistent subsets that cannot be eliminated by revising all the requirements that have been chosen for $P_\odot$.

Proposition 5.1. *Let $\Pi_\odot$ be the output of Algorithm 1 given R and S. Then $\forall P_\odot \in \Pi_\odot$, $P_\odot \preceq P'_\odot$ for any valid proposal $P'_\odot$ for handing inconsistency in R w.r.t. S.*

Proof. Suppose that $P_\odot \preceq P'_\odot$ does not hold, then $\exists M \in \mathsf{MI}(R \cup S_I)$, such that $\forall\alpha \in P_\odot \cap M$, $\exists\beta \in P'_\odot \cap M$ satisfies $\mathsf{BI}(R, \alpha|S) < \mathsf{BI}(R, \beta|S)$. In particular, consider $\alpha_0 \in \mathsf{BI}_{\mathsf{MAX}}(P_\odot \cap M)$. Obviously, $\exists\beta \in P'_\odot \cap M$ such that

$$\mathsf{BI}(R, \alpha_0|S) < \mathsf{BI}(R, \beta|S). \qquad (*)$$

On the other hand, let $P_{\alpha_0} = \{\gamma|\gamma \in P_\odot \setminus \{\alpha_0\}, \mathsf{BI}(R, \alpha_0|S) \leq \mathsf{BI}(R, \gamma|S)\}$. Furthermore, let $Q = \mathsf{MI}(R \cup S_I) \setminus \{M|M \in \mathsf{MI}(R \cup S_I), M \cap P_{\alpha_0} \neq \emptyset\}$, then $\alpha_0 \in \mathsf{BI}_{\mathsf{MAX}}(R \cap \bigcup Q)$. So, $\forall\beta \in M$,

$$\mathsf{BI}(R, \alpha_0|S) \geq \mathsf{BI}(R, \beta|S),$$

which contradicts $(*)$. □

Given a set of potential proposals for handling inconsistency in a requirements specification with regard to a given scenario, intuitively, the proposals that result in less incompleteness w.r.t. the given scenario are the ones with more preference. Let $\mathsf{CM}_{\mathsf{MIN}}(\Pi_\odot)$ be the set of the most preferred proposals in $\Pi_\odot$, i.e.,

$$\mathsf{CM}_{\mathsf{MIN}}(\Pi_\odot) = \{P_\odot \in \Pi_\odot|\forall P'_\odot \in \Pi_\odot, \mathsf{C}((R \setminus P_\odot)|S) \leq \mathsf{C}((R \setminus P'_\odot)|S)\}.$$

Essentially, $\mathsf{CM}_{\mathsf{MIN}}(\Pi_\odot)$ is exactly the set of the proposals in $\Pi_\odot$ that result in the lowest incompleteness in the revised requirements specification.

We propose an algorithm for generating proposals for handling incompleteness given a set of proposals for inconsistency handling, as described in Algorithm 2.

Algorithm 2. GENERATE_PROPOSALS_FOR_INCOMRE(S, R, $\Pi_{\odot}$)
Input: A Scenario S, the related requirements specification R, and a set of proposals for inconsistency handling $\Pi_{\odot}$
Output: A set of proposals $\Pi_{\odot\oplus}$

```
1 $\Pi_{\odot\oplus} \leftarrow \emptyset$
2 $\Pi_0 \leftarrow \emptyset$
3 $\Pi_0 \leftarrow \mathsf{CM}_{\mathsf{MIN}}(\Pi_{\odot})$
4 for each $P_{\odot} \in \Pi_0$
5     if $\mathsf{C}((R \setminus P_{\odot})|S) = 0$ then $P_{\oplus} = \emptyset$
6     else $P_{\oplus} = \{\alpha | \alpha \in S_E, (R \setminus P_{\odot}) \cup S_I \nvdash \alpha\}$
7     $P_{\odot\oplus} = \langle P_{\odot}, P_{\oplus} \rangle$
8     $\Pi_{\odot\oplus} \leftarrow \Pi_{\odot\oplus} \cup \{P_{\odot\oplus}\}$
9 return $\Pi_{\odot\oplus}$
```

Informally speaking, this algorithm aims to extend each of the most preferred proposals for handling inconsistency to a proposal for handling incompleteness due to inconsistency handling as well as inconsistency in R. The loop (Lines 4–8) ensures that $P_{\oplus}$ is exactly the set of expected responses that cannot be derived from $R \setminus P_{\odot}$ in S if $P_{\odot}$ is one of the most preferred proposals.

Recall the approaches to measuring redundancy, in which we focus on how to capture the most redundant perspective of a requirement specification w.r.t. a given scenario. Correspondingly, we need to identify the requirements with the maximal blame for the redundancy, i.e., the largest maximal potential redundancies. We propose an algorithm for identifying the most preferred proposals for handling redundancy given $\Pi_{\odot\oplus}$, as described in Algorithm 3.

Algorithm 3. GENERATE_PROPOSALS_FOR_REDUNDRE(S, R, $\Pi_{\odot\oplus}$)
Input: A Scenario S, the related requirements specification R, and a set of proposals for inconsistency and incompleteness handling $\Pi_{\odot\oplus}$
Output: A set of proposals $\Pi_{\odot\oplus\ominus}$

```
1  $\Pi_{\odot\oplus\ominus} \leftarrow \emptyset$
2  for each $P_{\odot\oplus} \in \Pi_{\odot\oplus}$
3      if $\mathsf{D}_{\mathsf{r}}(R \cup P_{\oplus} \setminus P_{\odot}|S) > 1$ then
4          for each $R' \in \mathsf{MR}_{\mathsf{MAX}}(R \cup P_{\oplus} - P_{\odot}|S)$
5              $P_{\ominus} = R'$
6              $P_{\odot\oplus\ominus} = \langle P_{\odot}, P_{\oplus}, P_{\ominus} \rangle$
7              $\Pi_{\odot\oplus\ominus} \leftarrow \Pi_{\odot\oplus\ominus} \cup \{P_{\odot\oplus\ominus}\}$
8      else $P_{\ominus} = \emptyset$
9          $P_{\odot\oplus\ominus} = \langle P_{\odot}, P_{\oplus}, P_{\ominus} \rangle$
10         $\Pi_{\odot\oplus\ominus} \leftarrow \Pi_{\odot\oplus\ominus} \cup \{P_{\odot\oplus\ominus}\}$
11 return $\Pi_{\odot\oplus\ominus}$
```

Based on the three base algorithms, we present an algorithm for generating the most appropriate proposals for handling non-canonical requirements as follows:

Algorithm 4. GENERATE_PROPOSALS_FOR_NONCANRE(S, R)
Input: A Scenario S and the related requirements R
Output: A set of proposals Π

```
1 $\Pi \leftarrow \emptyset$
2 $\Pi_{\odot} \leftarrow$ GENERATE_PROPOSALS_FOR_INCONRE($S$, $R$)
3 $\Pi_{\odot\oplus} \leftarrow$ GENERATE_PROPOSALS_FOR_INCOMRE($S$, $R$, $\Pi_{\odot}$)
4 $\Pi_{\odot\oplus\ominus} \leftarrow$ GENERATE_PROPOSALS_FOR_REDUNDRE($S$, $R$, $\Pi_{\odot\oplus}$)
5 $\Pi \leftarrow \Pi_{\odot\oplus\ominus}$
6 return $\Pi$
```

Now we use the following example to illustrate these algorithms.

Example 5.2. Consider Example 2.1 again. Consider a scenario $S = \langle S_I, S_E \rangle$, where

$$S_I = \{\mathtt{Eme(vis1)}, \mathtt{Aut(vis2)}, \neg\mathtt{Eme(vis2)}, \neg\mathtt{Aut(vis3)}, \neg\mathtt{Eme(vis3)}\},$$

$$S_E = \{\mathtt{Ent(vis1)}, \mathtt{Ent(vis2)}, \neg\mathtt{Ent(vis3)}, \mathtt{Ala(vis3)}, \neg\mathtt{Act(vis3, entr)}, \mathtt{Act(vis3, help)}\}.$$

Evidently,

$$R \cup S_I \vdash \texttt{Ent}(\texttt{vis1}) \wedge \neg\texttt{Ent}(\texttt{vis1}).$$

$$\mathsf{MI}(R \cup S_I) = \{\{(\forall \texttt{x})(\neg\texttt{Aut}(\texttt{x}) \to \neg\texttt{Ent}(\texttt{x})), (\forall \texttt{x})(\texttt{Eme}(\texttt{x}) \to \texttt{Ent}(\texttt{x}) \wedge \neg\texttt{Aut}(\texttt{x})), \texttt{Eme}(\texttt{vis1})\}\}.$$

Then

$$\mathsf{BI}((\forall \texttt{x})(\neg\texttt{Aut}(\texttt{x}) \to \neg\texttt{Ent}(\texttt{x}))|S) = \mathsf{BI}((\forall \texttt{x})(\texttt{Eme}(\texttt{x}) \to \texttt{Ent}(\texttt{x}) \wedge \neg\texttt{Aut}(\texttt{x}))|S) = \frac{1}{9}.$$

According to Algorithm 1, we can get

$$\Pi_{\odot} = \{P_{1\odot}, P_{2\odot}\},$$

where

$$P_{1\odot} = \{(\forall \texttt{x})(\neg\texttt{Aut}(\texttt{x}) \to \neg\texttt{Ent}(\texttt{x}))\},$$
$$P_{2\odot} = \{(\forall \texttt{x})(\texttt{Eme}(\texttt{x}) \to \texttt{Ent}(\texttt{x}) \wedge \neg\texttt{Aut}(\texttt{x}))\}.$$

Furthermore,

$$(R - P_{1\odot}) \cup S_I \nvdash \{\texttt{Act}(\texttt{vis3}, \texttt{help})\},$$
$$(R - P_{2\odot}) \cup S_I \nvdash \{\texttt{Ent}(\texttt{vis1}), \texttt{Act}(\texttt{vis3}, \texttt{help})\}$$

Then

$$\mathsf{C}((R \setminus P_{1\odot})|S) = \frac{1}{6} < \frac{1}{3} = \mathsf{C}((R \setminus P_{2\odot})|S),$$

and

$$\mathsf{CM}_{\mathsf{MIN}}(\Pi_{\odot}) = \{P_{1\odot}\}.$$

So according to Algorithm 2, we can get

$$\Pi_{\odot\oplus} = \{P_{1\odot\oplus} = \langle P_{1\odot}, P_{1\oplus}\rangle\},$$

where

$$P_{1\oplus} = \{\texttt{Act}(\texttt{vis3}, \texttt{help})\}.$$

Finally,

$$\mathsf{D_r}(R \cup P_{1\oplus} \setminus P_{1\odot}|S) > 1$$

and

$$(R \cup P_{1\oplus} \setminus P_{1\odot}) \setminus \{(\forall \texttt{x})(\texttt{Ala}(\texttt{x}) \to \neg\texttt{Act}(\texttt{x}, \texttt{entr}))\} \cup \texttt{S}_{\texttt{I}} \vdash \texttt{S}_{\texttt{E}},$$
$$(R \cup P_{1\oplus} \setminus P_{1\odot}) \setminus \{(\forall \texttt{x})(\neg\texttt{Aut}(\texttt{x}) \wedge \neg\texttt{Eme}(\texttt{x}) \to \neg\texttt{Act}(\texttt{x}, \texttt{entr}))\} \cup \texttt{S}_{\texttt{I}} \vdash \texttt{S}_{\texttt{E}}.$$

And

$$\mathsf{MR}_{\mathsf{MAX}}(R \cup P_{1\oplus} \setminus P_{1\odot}|S) = \{R_1, R_2\},$$

where

$$R_1 = \{(\forall x)(\mathtt{Ala}(x) \to \neg\mathtt{Act}(x, \mathtt{entr}))\},$$
$$R_2 = \{(\forall x)(\neg\mathtt{Aut}(x) \wedge \neg\mathtt{Eme}(x) \to \neg\mathtt{Act}(x, \mathtt{entr}))\}.$$

So according to Algorithm 3, we can get the valid potential proposal for non-canonical requirements w.r.t. S as follows:

$$\Pi = \Pi_{\odot\oplus\ominus} = \{P_{1\odot\oplus\ominus} = \langle P_{1\odot}, P_{1\oplus}, P_{1\ominus}\rangle, \quad P'_{1\odot\oplus\ominus} = \langle P_{1\odot}, P_{1\oplus}, P'_{1\ominus}\rangle\},$$

where

$$P_{1\ominus} = \{(\forall x)(\mathtt{Ala}(x) \to \neg\mathtt{Act}(x, \mathtt{entr}))\},$$
$$P'_{1\ominus} = \{(\forall x)(\neg\mathtt{Aut}(x) \wedge \neg\mathtt{Eme}(x) \to \neg\mathtt{Act}(x, \mathtt{entr}))\}.$$

That is, there are two possible ways of handling non-canonical requirements w.r.t. S as follows:

- *Proposal I*:
  (I1) To revise (r1);
  (I2) To add requirements information about activating the button for help;
  (I3) To suspend or revise (r4).
- *Proposal II*:
  (II1) To revise (r1);
  (II2) To add requirements information about activating the button for help;
  (II3) To suspend or revise (r7).

Note that the main difference between Proposal I and Proposal II is that they choose different requirements to be suspended. The two proposals agree with each other in both the requirements to be revised and the expected responses to be included. Actually, both proposals are intuitive for the following reasons:

- Although both (r1) and (r5) are involved in the inconsistency, only (r5) describes correctly the requirement about emergency ones in this scenario, then it is intuitive to consider (r1) as the requirement to be revised.
- Evidently, the information about activating the button for help is missing in the requirements specification for the given scenario, so it is also natural to consider $\{\mathtt{Act}(\mathtt{vis3}, \mathtt{help})\}$ as $P_{\oplus}$ in any feasible proposal.
- Note that if we keep (r7) in the requirements specification, then (r4) will be redundant. Alternatively, if we keep (r4) in the requirements specification, then (r7) will be redundant. Therefore, there are two possible ways of eliminating redundancy.

Note that these algorithms aim to generate potential proposals appropriate only for handling non-canonical requirements with regard to a given scenario rather than for revising the whole requirements specification directly. However, these algorithms can provide practical support for revising the whole requirements specification for several reasons:

- Note that local consistency in a set of requirements with regard to a given scenario does not guarantee the global consistency of a requirements specification. But inconsistency in a set of requirements w.r.t. a scenario definitely shows that the whole requirements specification is inconsistent. Actually, as mentioned in [5], it is particularly difficult to check the global consistency of a large requirements specification. When checking consistency of the whole requirements specification becomes infeasible, identifying and handling inconsistency based on scenarios may be considered as a practical way of revising the whole requirements specification as much as possible.
- Similar to the relation between local consistency and global consistency, completeness of a set of requirements w.r.t. a scenario does not guarantee completeness of the whole requirements specification. But local incompleteness of a set of requirements w.r.t. a scenario is meaningful for verifing incompleteness of the whole requirements specification. Informally speaking, each potential proposal for handling local incompleteness provides a chance to elicit the requirements information about certain application setting. On the other hand, checking completeness of requirements is also particularly difficult in many cases [12]. Moreover, the development of any complex system inevitably involve many stakeholders, and each stakeholder is familiar with only some special application scenarios. This makes scenario-based incompleteness handling even more necessary.
- Note that potential proposals for handling redundancy in a set of requirements w.r.t. a scenario are concentrated on suspending redundant requirements w.r.t. a scenario. It also provides a chance to scan and confirm these requirements and to improve the quality of the whole requirements specification.
- Essentially, the logical framework presented above aims to test and revise requirements at the requirements stage in some sense if we consider scenarios as prototypes of conceptual test cases. Similar to software testing, the testing

and revision are often built upon a family of scenarios. In this sense, the algorithm provides support for requirements testing based on each scenario.

Note that the potential proposals generated by these algorithms aim to simply point out which requirements may be considered to be revised rather than how to revise them in the given scenario. Obviously, the choice of scenarios also has impact on decision making on handling non-canonical requirements. However, how to design scenarios appropriate for checking and handling non-canonical requirements is a rather context-sensitive issue. It is difficult to provide an universal way of design or choose scenarios in general. Whatever actions developers choose according to these proposals, the revised requirements need to be checked again within the logical framework presented in this paper.

6. Related work

Handling non-canonical requirements is a common issue in requirements engineering. Most logic-based techniques for managing non-canonical requirements only handle inconsistency in requirements. Relatively few logic-based approaches to managing non-canonical requirements have been proposed along with inconsistency handling, especially to making appropriate proposals for handling non-canonical requirements. In the following, we compare our approach with some of closely related approaches.

Our previous paper [13] may be considered as an attempt of managing non-canonical requirements by extending proposals for handling inconsistency in requirements, which focuses on how to formulate non-canonical requirements rather than how to identify and generate proposals for handling non-canonical requirements practically. In contrast, the measure-driven framework presented in this paper focuses on how to identify and generate appropriate proposals for handling non-canonical requirements in a feasible way. In particular, a family of measures for non-canonical requirements play an important role in generating more appropriate proposals for handling non-canonical requirements.

In addition, the interplay between inconsistency and incompleteness in requirements specifications has been considered in handling inconsistent requirements specifications in [12,11]. However, resolving inconsistency in a requirements specification cannot ensure that the revised requirements specification does not contain redundant information. Given a scenario, a requirements specification may contain redundant information for some expected behaviors of the system-to-be in the scenario, even if the specification lacks information for other expected behaviors of the system-to-be. That is, redundancy may exist along with incompleteness in a requirements specification. Compared to [11,12], this paper has considered redundancy as well as inconsistency and incompleteness. That is, we focus on monitoring redundancy as well as incompleteness resulted from inconsistency handling.

Note that potential proposals for handling incompleteness focus on pointing out which expected responses cannot be derived from requirements in a scenario. That is, we consider the set of missing expected responses as a hint on eliciting new requirements about the scenario. However, to be more practical, we may consider using abduction techniques presented in [26] in handling incompleteness as well as handling inconsistency. This is one of the main directions of our future work.

On the other hand, our previous papers [9,21,10] presented two approaches to identifying acceptable common proposals for handling inconsistency in multi-perspective requirements with qualitative priority levels (such as high and low). These approaches emphasize the importance of group decision making mechanisms such as negotiation among multiple perspectives [9,10] and combinatorial vote [21] in identifying requirements to be changed. In contrast, the framework presented in this paper aims not only to resolve inconsistency, but also to evaluate the impact of proposals for inconsistency handling on incompleteness or redundancy in a requirements specification with regard to a given scenario. Moreover, the framework presented in this paper emphasizes the role of appropriate measurement of non-canonical requirements in identifying requirements to be changed. Actually, it is not hard to adapt our framework to requirements with priority levels if we adopt the corresponding measures for inconsistent prioritized knowledge bases such as [27].

With regard to the potential implementation of the proposed measures in this paper, the core of inconsistency measures is to compute minimal inconsistent subsets for a knowledge base. One of the underlying problems is to check whether a set of formulas is consistent, i.e., a SAT problem. On the other hand, the problem of incompleteness and that of redundancy are tightly associated with the problem of inconsistency, since $K \cup \{\neg\alpha\} \nvdash \bot$ if $K \nvdash \alpha$, and $K \cup \{\neg\alpha\} \vdash \bot$ if $K \nvdash \bot$ and $K \vdash \alpha$ for all satisfiable formula α. It is promising to transfer the computation of measures for incompleteness and redundancy to the SAT-based computation. This makes the core computationally hard, since SAT is NP-complete [28], and checking whether a set of clauses is a minimal inconsistent subset is DP-complete [29]. However, as pointed out in [17], the impressive progress in SAT solvers in recent years has promoted techniques for practically identifying minimal inconsistent subsets of a knowledge base. Some algorithms such as [30] have been proposed to practically find each minimal inconsistent subset (called Minimally Unsatisfiable Subformulas or MUS in these algorithms) of a knowledge base. On the other hand, as an efficient Java library of SAT solvers, SAT4J library 2[4] provides the first-time users of SAT with black boxes, who want to incorporate SAT technologies into their application without worrying about the details.[5] The implementation of a prototype tool with the SAT4J library as well as the existing algorithm presented in [30] is one of the main directions of future work.

[4] https://wiki.objectweb.org/sat4j/.
[5] https://www.sat4j.org.

7. Conclusions

It has increasingly been recognized that inconsistency handling in requirements engineering is not an isolated problem. It always results in some other undesirable issues such as incompleteness and redundancy of requirements. Therefore, it is intuitive to extend logic-based approaches to manage non-canonical requirements such as incompleteness and redundancy in requirements engineering. This paper has made the following contributions in managing non-canonical requirements in requirements engineering:

(a) We have proposed a logical framework for managing non-canonical requirements, including inconsistent, incomplete, and redundant requirements, which consists of five main parts, identifying non-canonical requirements, measuring non-canonical requirements, generating candidate proposals for handling non-canonical requirements, choosing commonly acceptable proposals and revising non-canonical requirements according to the chosen proposals.
(b) We have presented a family of measures for non-canonical requirements, including ones for inconsistency, incompleteness, and redundancy.
(c) We have proposed three algorithms for generating proposals for handling non-canonical requirements.

Note that the logical framework is based on the choice of scenarios in some sense. Moreover, we assume that all the scenarios are developed correctly within the logical framework. However, how to develop and choose scenarios appropriately is still a challenge in requirements specification. Some scenarios may also be problematic, and even contradict each other. Moreover, proposals for handling incompleteness use only expected responses that cannot be derived in a scenario as a hint on eliciting missing requirements. Such proposals are sketchy in some cases when we consider the practicability of proposals. In future work, we will adapt the logical framework for revising scenarios as well as requirements in requirements engineering practice.

Acknowledgements

This work was partly supported by the National Natural Science Foundation of China under Grant No. 61170300, the National Basic Research 973 program of China under Grant No. 2009CB320701, and the Key Project of National Natural Science Foundation of China under Grant No. 90818026, and the NSFC and the British Royal Society China-UK Joint Project.

References

[1] V. Gervasi, D. Zowghi, Reasoning about inconsistencies in natural language requirements, ACM Transactions on Software Engineering and Methodologies 14 (3) (2005) 277–330.
[2] A. Hunter, B. Nuseibeh, Managing inconsistent specifications: reasoning, analysis, and action, ACM Transactions on Software Engineering and Methodology 7 (4) (1998) 335–367.
[3] A. Lamsweerde, R. Darimont, E. Letier, Managing conflicts in goal-driven requirements engineering, IEEE Transactions on Software Engineering 24 (11) (1998) 908–926.
[4] S. Easterbrook, M. Chechik, A framework for multi-valued reasoning over inconsistent viewpoints, in: A.H. Müller, M.J. Harrold, and W. Schäfer (Eds.), Proceedings of International Conference on Software Engineering (ICSE'01), Toronto, Canada, 2001, pp. 411–420.
[5] B. Nuseibeh, S. Easterbrook, A. Russo, Leveraging inconsistency in software development, IEEE Computer 33 (4) (2000) 24–29.
[6] B. Nuseibeh, S. Easterbrook, A. Russo, Making inconsistency respectable in software development, Journal of Systems and Software 58 (2) (2001) 171–180.
[7] S. Easterbrook, M. Chechik, 2nd international workshop on living with inconsistency, Software Engineering Notes 26 (6) (2001) 76–78.
[8] A. Martinez, J. Arias, A. Vilas, Merging requirements views with incompleteness and inconsistency, Proceedings of Australian Software Engineering Conference 2005, IEEE Computer Society, Los Alamitos, CA, 2005, pp. 58–67.
[9] K. Mu, Z. Jin, D. Zowghi, A priority-based negotiations approach for handling inconsistency in multi-perspective software requirements, Journal of Systems Science and Complexity 21 (4) (2008) 574–596.
[10] K. Mu, W. Liu, Z. Jin, A. Yue, R. Lu, D. Bell, Handling inconsistency in distributed software requirements specifications based on prioritized merging, Fundamenta Informaticae 91 (3–4) (2009) 631–670.
[11] A.B.B. Martinez, J.J.P. Arias, A.F. Vilas, J.G. Duque, M.L. Norse, R.P.D. Redondo, Y.B. Fernandez, On the interplay between inconsistency and incompleteness in multi-perspective requirements specifications, Information and Software Technology 50 (4) (2008) 296–321.
[12] D. Zowghi, V. Gervasi, On the interplay between consistency, completeness, and correctness in requirements evolution, Information and Software Technology 45 (14) (2003) 993–1009.
[13] K. Mu, Z. Jin, R. Lu, Y. Peng, Handling non-canonical software requirements based on annotated predicate calculus, Knowledge and Information Systems 11 (1) (2007) 85–104.
[14] K. Mu, Q. Zhang, Z. Jin, Verifying software requirements based on answer set programming, in: D. Karagiannis, Z. Jin (Eds.), KSEM2009, LNAI, vol. 5914, Springer, 2009, pp. 263–274.
[15] K. Mu, Z. Jin, R. Lu, W. Liu, Measuring inconsistency in requirements specifications, in: L. Godo (Ed.), Proc. ECSQARU2005, LNCS, vol. 3571, Springer Verlag, New York, Secaucus, NJ, USA, 2005, pp. 440–451.
[16] D. Jackson, Automating frist-order relational logic, ACM SIGSOFT Software Engineering Notes 25 (6) (2000) 130–139.
[17] A. Hunter, S. Konieczny, Measuring inconsistency through minimal inconsistent sets, in: G. Brewka, J. Lang (Eds.), Principles of knowledge representation and reasoning: proceedings of the eleventh international conference (KR08), 2008, pp. 358–366.
[18] D. Gabbay, A. Hunter, Making inconsistency respectable 2: meta-level handling of inconsistent data, in: M. Clarke, R. Kruse, S. Moral (Eds.), Symbolic and Quantitative Approaches to Reasoning and Uncertainty, European Conference, ECSQARU'93, Granada, Spain, November 8-10, 1993, Proceedings, Lecture Notes in Computer Science, vol. 747, 1993, pp. 129–136.
[19] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, B. Nuseibeh, Inconsistency handling in multi-perspective specifications, IEEE Transactions on Software Engineering 20 (8) (1994) 569–578.
[20] K. Mu, W. Liu, Z. Jin, A general framework for measuring inconsistency through minimal inconsistent sets, Knowledge and Information Systems 27 (1) (2011) 85–114.

[21] K. Mu, Z. Jin, Identifying acceptable common proposals for handling inconsistent software requirements, in: J. Derrick, J. Vain (Eds.), Forte 2007, LNCS, vol. 4574, 2007, pp. 296–308.
[22] A. Hunter, S. Konieczny, Shapley inconsistency values, in: P. Doherty, J. Mylopoulos, C. Welty (Eds.), Principles of knowledge representation and reasoning: proceedings of the 10th international conference (KR06), AAAI Press, 2006, pp. 249–259.
[23] R. Reiter, A theory of diagnosis from first priniciples, Artificial Intelligence 32 (1) (1987) 57–95.
[24] K. Knight, Measuring inconsistency, Journal of Philosophical Logic 31 (1) (2002) 77–98.
[25] M. Thimm, Measuring inconsistency in probabilistic knowledge bases, in: J. Bilmes, Y. Andrew (Eds.), Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI'09), Montreal, Canada, June 2009, AUAI Press, 2009, pp. 530–537.
[26] A. Russo, R. Miller, B. Nuseibeh, J. Kramer, An abductive approach for analysing event-based requirements specifications, in: P.J. Stuckey (Ed.), Logic Programming, 18th International Conference, ICLP 2002, Proceedings, Lecture Notes in Computer Science, vol. 2401, Springer, Copenhagen, Denmark, 2002, pp. 22–37.
[27] K. Mu, W. Liu, Z. Jin, Measuring the blame of each formula for inconsistent prioritized knowledge bases, Journal of Logic and Computation 22 (3) (2012) 481–516.
[28] S.A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the Third Annual ACM Symposium on the Theory of Computing, New York, 1977, pp. 151–158.
[29] T. Eiter, G. Gottlob, On the complexity of propositional knowledge base revision, updates, and counterfactuals, Artificial Intelligence 57 (2–3) (1992) 227–270.
[30] É. Grégoire, B. Mazure, C. Piette, Using local search to find MSSes and MUSes, European Journal of Operational Research 199 (3) (2009) 640–646.